Dr. Dobb's Journal

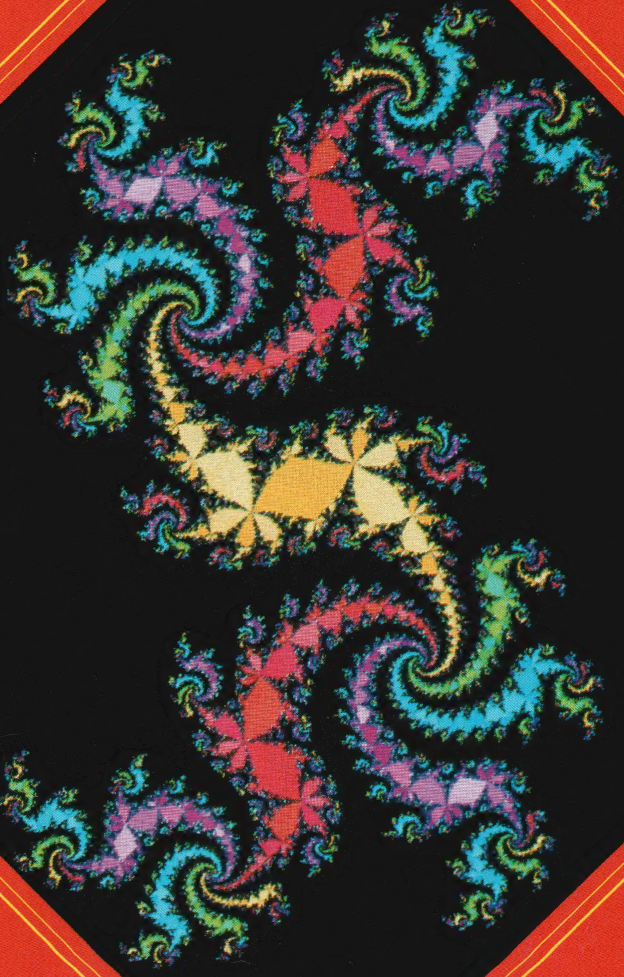
SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

#103 May 1985 $2.95 (3.50 Canada)

Graphics Algorithms

A C Command-line Processor

PC/AT & Mac Notes & 68K Math



Using Prolog

Testing C/80 & Better Basic

Speed, Power, Price.
Borland's Turbo Pascal Family.

$69.95

The industry standard. With more than 250,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use!

Jeff Duntemann, PC Magazine: *"Language deal of the century . . . Turbo Pascal: It introduces a new programming environment and runs like magic."*

Dave Garland, Popular Computing: *"Most Pascal compilers barely fit on a disk, but Turbo Pascal packs an editor, compiler, linker, and run-time library into just 29K bytes of random-access memory."*

Jerry Pournelle, BYTE: *"What I think the computer industry is headed for: well documented, standard, plenty of good features, and a reasonable price."*

Portability. Turbo Pascal is available today for most computers running PC DOS, MS DOS, CP/M 80 or CP/M 86. A XENIX verison of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

High resolution monochrome graphics for the IBM PC and the Zenith 100 computers

$54.95

NEW

Dazzling graphics and painless windows. The Turbo Graphix Toolbox will give even a beginning programmer the expert's edge. It's a complete library of Pascal procedures that include:

—Full graphics window management.
—Tools that will allow you to draw and hatch pie charts, bar charts, circles, rectangles and a full range of geometric shapes.
—Procedures that will save and restore graphic images to and from disk.
—Functions that will allow you to precisely plot curves.
—Tools that will allow you to create animation or solve those difficult curve fitting problems.
and much, much more . . . . .

No sweat and no royalties. You may incorporate part, or all of these tools in your programs, and yet, we won't charge you any royalties. Best of all, these functions and procedures come complete with commented source code on disk ready to compile!

Searching and sorting made simple

$54.95

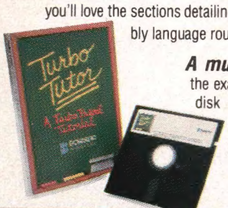
The perfect complement to Turbo Pascal. It contains: ***Turbo-Access***, a powerful implementation of the state-of-the-art B+tree ISAM technique; ***Turbo-Sort***, a super efficient implementation of the fastest data sorting algorithm, "Quicksort on disk". And much more.

Jerry Pournelle, BYTE: "The tools include a B+tree search and a sorting system; I've seen stuff like this, but not as well thought out, sell for hundreds of dollars."

Get started right away: free database! Included on every Toolbox disk is the source code to a working data base which demonstrates how powerful and easy to use the Turbo-Access system really is. Modify it to suit your individual needs or just compile it and run.

Remember, no royalties!

From Start to Finish in 300 pages. Turbo Tutor is for everyone, from novice to expert. Even if you've never programmed before, Turbo Tutor will get you started right away. If you already have some experience with Pascal or another programming language, Turbo Tutor will take you step by step through topics like data structures and pointers. If you're an expert, you'll love the sections detailing subjects such as "how to use assembly language routines with your Turbo Pascal programs."



A must. You'll find the source code for all the examples in the book on the accompanying disk ready to compile. Turbo Tutor might be the only reference on Pascal and programming you'll ever need.

$34.95

NOT COPY-PROTECTED

TURBO PASCAL FAMILY

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS
___ CP/M 80 ___ CP/M 86
My computers' name/model is: ____________

The disk size I use is:
☐ 3½" ☐ 5¼" ☐ 8"

Pascal 3.0	$ 69.95	____
Pascal w/8087	$109.90	____
Pascal w/BCD	$109.90	____
Pascal w/8087 & BCD	$124.95	____
Turbo Toolbox	$ 54.95	____
Turbo Graphix	$ 54.95	____
Turbo Tutor	$ 34.95	____

*These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Amount: (CA 6% tax) ____________
Payment: VISA MC BankDraft Check
Credit Card Expir. Date: ____ / ____
Card #: | | | | | | | | | | | | | | | |

Name: ____________
Shipping Address: ____________
____
City: ____________
State: ________ Zip: ________
Telephone: ____________

F11

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. … USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

BORLAND INTERNATIONAL

Software's Newest Direction
4585 Scotts Valley Drive
Scotts Valley, CA 95066
TELEX 172373

Circle ***no. 11*** *on reader service card.*

Up Your AT™ for $56 per Megabyte!

■ **While our specifications certainly speak for themselves, we thought you still might like to hear from some of our users:**

■ "Emerald Systems expands the potential of PCs by providing the ability to access large amounts of data on line, quickly and reliably."
Terry Baptiste, Computerland, Lafayette, Ca.

■ "Service and support is great, which is an unusual experience. Emerald's software for backup and restore is invaluable. Can't put a price on it. Productivity and efficiency has increased at least 50%!"
Bruce Kittinger, Pinon Systems, Ft. Collins, Co.

■ Runs like a champ with 3-Com Ethernet."
Alvaro Ramirez, Micro Age, Miami, Fl.

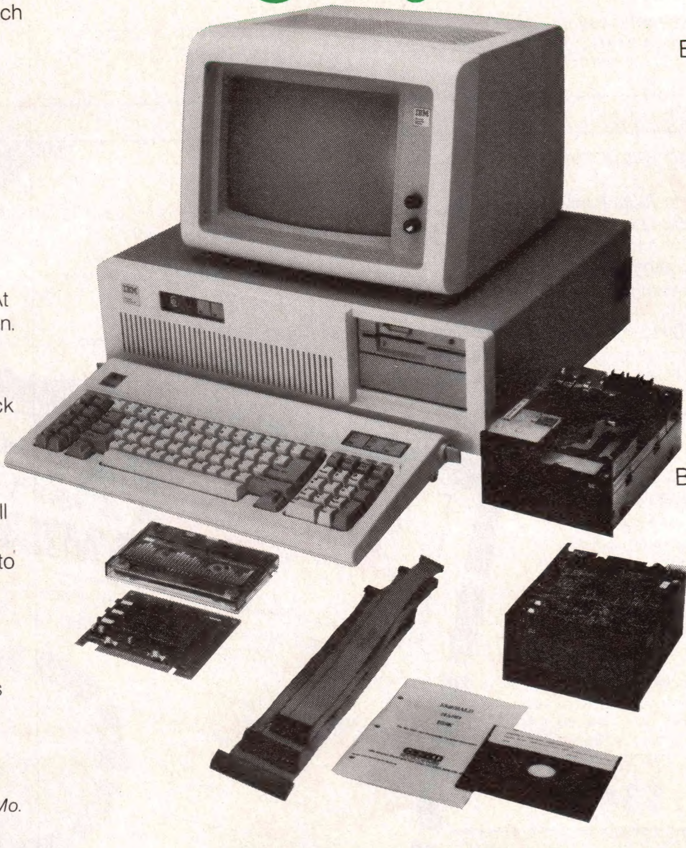
■ ". . . high capacity and flexibility. At last, a tape back up we can count on. And the price is right!"
John Acres, EDT, Las Vegas, Nv.

■ "The speed at which you can back up is very impressive."
Jim McEwen, Mercy Hospital, Portland, Me.

■ "When Emerald says your unit will be there on Thursday, it's there on Thursday! Delighted we were able to exceed the usual 32 megabyte restriction."
Steven Mayer, Take One Company, New York, N.Y.

■ "The Emerald 70 MB hard disk is extremely easy to install and work with. Emerald has a complicated piece of equipment made easy to use."
Tom Edler, Jewish Hospital, St. Louis, Mo.

■ "Our Emerald fixed disk installed quickly and easily. Emerald's reliable disk and tape backup further enhances LIBRA's high function accounting software."
Kenn White, Libra Programming, Salt Lake City, Ut.



HARD DISK

We've broken through the 32 MByte DOS barrier!

Now you can create up to 240 MByte databases on one file, with multiple volumes per disk drive. You get 14 times more storage with a 30% increase in access speed!

And that's not all:

Expands up to 280 MB for as little as $56 per megabyte.
6 expansion slots
Up to 24 volumes
Drive sizes: 40,70,140
Internal or External
Also for PC,XT, and compatibles
Supports all PC compatible networks
Simple menu-driven installation

TAPE BACKUP

¼" cartridge
½" 9 track
60 Megabytes
No lost data from tape run out
Backup and Restore Utility (BRU™) software included
LAN Compatible*

***FREE APPLICATION GUIDE:** "IBM LAN Installation and Implementation."

Call (619) 270-1994, or write to

Emerald™ Systems Corporation

Mainframe Storage for Micros
4901 Morena Blvd,
San Diego, 92117
TLX 323458 EMERSYS
EASYLINK 62853804

Distributed by Manchester Equipment of NYC, and selected Entre, MicroAge and Computerland stores.

Emerald, BRU, Up Your AT, and Mainframe Storage for Micros are registered trademarks of Emerald Systems Corporation. IBM PC/XT/AT are registered trademarks of IBM Corporation.

EMERALD™
SYSTEMS CORPORATION
Mainframe Storage for Micros™

Circle **no. 32** on reader service card.

Dr. Dobb's Journal (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, 2464 Embarcadero Way, Palo Alto, CA 94303. **ISSN 0278-6508**

Subscription Rates: $25 per year within the United States, $44 for first class to Canada and Mexico, $62 for airmail to other countries. Payment must be in U.S. Dollars, drawn on a U.S. Bank. For subscription problems, call: outside of CA 1-800-321-3333; within CA 1-619-485-6535.

Foreign Distributors: ASCII Publishing, Inc. (Japan), Computer Services (Australia), Computer Store (New Zealand), Computercollectief (Nederland), Homecomputer Vertriebs GMBH (West Germany), International Presse (West Germany), La Nacelle Bookstore (France), McGill's News Agency PTY LTD (Australia), Progresco (France).

People's Computer Company

Dr. Dobb's Journal is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a non-profit, educational corporation.

May 1985
Volume 10, Issue 5

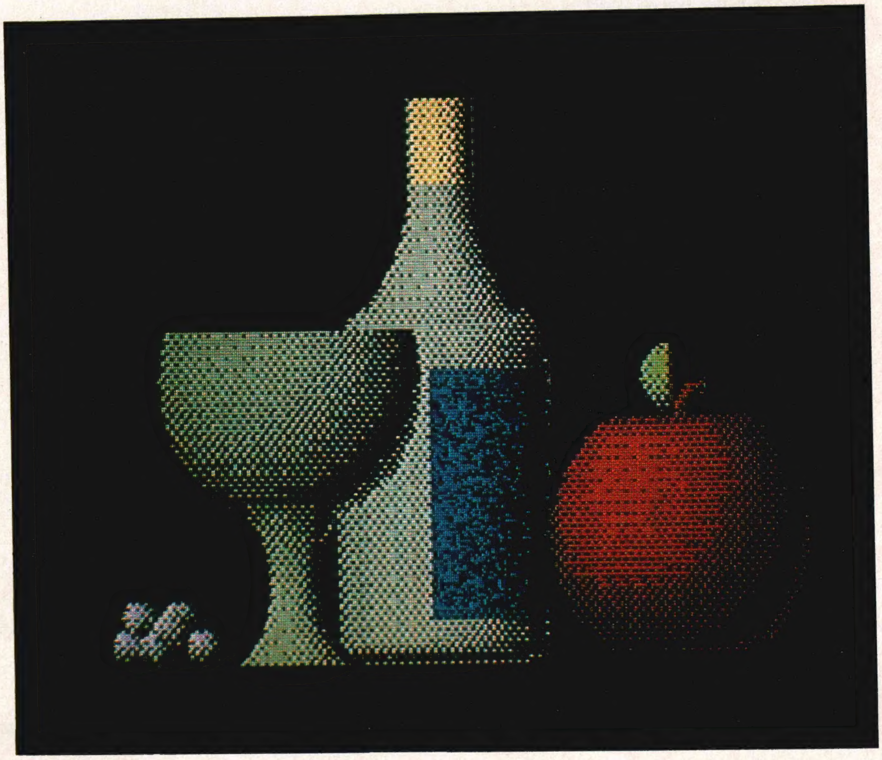
CONTENTS

Cover Artist

Our cover "Fractal Dragon," is the creation of Benoit Mandelbrot, Professor of the Practice of Mathematics at Harvard University. The most conspicuous characteristic of this dragon is that the eye and the imagination distinguish in it, in addition to the overall shape, an infinity of smaller features of every size down to the infinitesimal. For geometric shapes of this kind, Professor Mandelbrot coined the term "fractal." "Fractal Dragon" is a portion of "attractor" that occurs in a problem of dynamics. The cover illustration is a reproduction of Plate C5 of Mandelbrot's book *The Fractal Geometry of Nature* (Freeman 1982), drawn using computer programs by Mark R. Laff and V. Alan Norton. Copyright 1982 by B. B. Mandelbrot.

Referees Who Assisted With This Issue

Ted Carnevale, State University of New York at Stony Brook
Allen Holub, *DDJ* Contributing Editor
Robert Tripp, Editor-in-Chief of the late *MICRO*
James Woomer, Commodore-64 Guru and Gazebo Builder



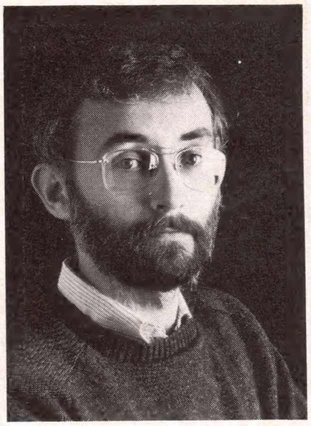
See Richard Rylander's "Solid Shape Drawing on the Commodore 64" page 50.

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

ARTICLES

COLUMNS

DEPARTMENTS



I want to comment on what others have been saying about two subjects germane to this or any issue of *DDJ*: graphics and copy-protected disks.

Frank Gaude regards the iconic interface of the Mac and its imitators as a regression to preliterate cave-scratchings. In his *Z-News* he says, "Long ago we used hieroglyphs to communicate. Then after a struggle an alphabet developed. Now we go back to pictures: icons. How many glyphs can we remember?"

In the macaphony of the current iconolatry this cautionary message is worthy of attention, but I suggest that it's not the whole story. The function of software is to realize metaphors. Icons are capable of doing that well but are no help in stepping outside existing metaphors to invent new ones; no help, that is, in developing new software. I would agree that icons are mnemonic for users and superfluous for programmers and let it go at that, but for the fact that I have recently seen a prototype of a system that takes icons a step further, a system in which actions can be assigned user-created visual tokens, and can be combined in ways that visually and kinaesthetically convey the form of the combination. Active, growing icons.

Furthermore, while the Mac icons are static tokens rather than elements of a true language, that is not true of all computer graphic aids to communication. French researchers are currently developing the means for transmitting sign language over telephone lines at commercial data rates. Their idea is to extract from a sequence of video frames just the essential components of a gesture and pass those along. This does not, I should make clear, mean matching the gesture to some template from a database of acceptable gestures; rather it means producing a real-time cartoon version of the gesturing party. The cartoons, subject to idiomatic and dialectical variations, are truly linguistic graphic elements, which is just what icons aren't.

As for copy-protected disks . . . .

". . . imagine yourself buying a new car. You have narrowed your choice to two cars, identical in all respects except that one will travel 100 mph and the other has a governor on it and cannot go past 55. Which car would you buy?"—Michael D. Brown, Central Point Software, producer of Copy II.

"People can rest assured that...we will always offer versions of our products that are not copy-protected."—Philippe Kahn, Borland International.

"Companies like Multimate and Borland have the right idea. Lotus, Ashton-Tate and MicroPro are the bad guys."—Edward Messerly, SF regional administrator of the GSA.

". . . a growing number of publishers are dropping traditional copy-protection methods, which they conclude have created ill-will and inconvenience for users . . . . Among the publishers who have abandoned copy-protection schemes is MicroPro . . . . Many software publishers, however, hope to see new protection methods . . . ."—Time/Life Access:Apple.

"These people are nutty."—Edward Messerly.

Yes. Something is nutty when a vendor can brag about also selling non-broken goods. It's really simple, isn't it? Vendors of software who want to stay vendors of software must find ways to profit from their efforts without breaking the tools they sell. The ability to copy files and disks is a facility I bought with my hardware and operating system. To try to sell me a program designed to subvert my system and hamper its operation is to try to enlist me in vandalism against my own property. Even granting that copy-crippling is a morally defensible policy, how could anyone believe that it was commercially viable in the long run? Nutty.

Michael Swaine

Michael Swaine

Another in a series of productivity notes on MS-DOS™ software from UniPress.

TEXT EDITING

UNIPRESS EMACS™

Subject: Multi-window full screen editor.

Multiple windows allow several files (or portions of the same file) to be edited simultaneously. Programmable through macros and the built-in compiled MLISP™ extension language.

Features:

- *Famed Gosling Version.*
- *Extensible through the built-in MLISP programming language and macros.*
- *Dozens of source code MLISP functions; including C, Pascal and MLISP syntax checking.*
- *EMACS runs on TI-PC™, IBM-PC AT,™ DEC Rainbow™ or any other MS-DOS machine. Requires at least 384k.*
- *Run Lattice® C or PsMake™ in the background and EMACS will point to any errors for ease of debugging. PsMake is a UNIX™-style "make" utility to automate the process of building complex programs.*
- *Optional Carousel Tools: UNIX-like facilities including cat, cp, cd, logout, ls, mv, pwd, rm, set, sh and more.*

Price:

EMACS	$475
One month trial	75
Available for UNIX and VMS.	
PsMake	179
Carousel Tools	149
Full System	1,299
Includes EMACS (object), PsMake, Lattice C, PHACT™ ISAM and Carousel Tools.	

COMPILER FOR THE 8086™ FAMILY

LATTICE® C COMPILER

Subject: Compiler for MS-DOS.

Lattice C Compiler is regarded as the finest compiler for MS-DOS and is running on thousands of 8086 systems.

Features:

- *Runs on the IBM-PC™ under MS-DOS 1.0, 2.0 or 3.0*
- *Produces highly optimized code.*
- *Small, medium, compact and large address models available.*
- *Standard C library.*
- *PLINK–optional full function linkage editor including overlay and support.*

Price:

Lattice C Compiler	$425
PLINK	425

ISAM FILE SYSTEM & REPORT WRITER

PHACT™

Subject: Powerful Keyed File Access for MS-DOS.

PHACT ISAM is a keyed B + tree file manager providing easy access to and manipulation of records in a database.

Features:

- *Supports fixed and variable length records (1-9999 bytes).*
- *Up to 9 alternate indices are supported.*
- *Record locking allows each record in the database to allow multiple simultaneous updates.*
- *Records can be accessed on full or partial key.*
- *Includes full Lattice linkable library and high-level functions.*
- *Optional, PHACT-rg, a powerful and flexible report generator which provides a high level, easy-to-use command language for formatting reports from existing PHACT databases. Available for UNIX, MS-DOS and VMS.*

Price:

PHACT ISAM	$250
PHACT-rg	165
Source Code available, call for terms.	

For more information on these and other UNIX software products, call or write: UniPress Software, Inc., 2025 Lincoln Hwy., Edison, NJ 08817. Telephone: (201) 985-8000. Order Desk: (800) 222-0550 (Outside NJ). Telex: 709418. Japanese Distributor: Softec 0480 (85) 6565. European Distributor: Modulator SA (031) 59 22 22.

OEM terms available. Mastercard/Visa accepted.

UniPress Software

Your Leading Source for UNIX Software.

Circle **no. 77** on reader service card.

Anorexia Cured

Dear *DDJ*:
I read with some interest the article 'Fatten Your Mac' that appeared in your January [1985, #99] issue and forthwith did a fattening on my Macintosh. There are a number of points that I would like to add to the article, however. Firstly, a fine supplier of the 256K DRAM chips was omitted. I refer to Microprocessors Unlimited, of Beggs, Oklahoma, (918) 267-4961. I was able to phone them late Friday, and the chips they sent out on Monday arrived here (in Seattle) on Tuesday. I was quite pleased with the price and the service. They accept Visa, and they even sent a brief note explaining proper handling of the chips.

Secondly, it is *not* necessary, though it may be desirable, to destroy the 64K DRAM chips. I desoldered the original chips with an Ungar model 2000 desoldering tool, and was able to remove the solder, let the board cool, then pry the chips from the motherboard, all without clipping a single lead. This damaged two printed circuit traces (pulled them off in the prying process); as the article recommends, *there is a safer way to proceed*. My technique cannot be advised in the absence of an excellent desoldering tool, or for those not confident in their abilitites to make printed circuit repairs. The article had so fired my enthusiasm that I yanked out the 64K DRAMs days before the 256K DRAM circuits arrived. I was able to verify the repairs, and the removed memory chips, by installing sockets, plugging the 64K DRAMs into those sockets, and using the Macintosh. The nature of the lifted printed circuit traces (they showed copper color on the ICs to which the traces stuck) made detection easy, and no faults were missed. The board worked on the first try.

Thirdly, the multiplexer can be installed either piggybacked on another chip, or can be placed on a separate board, attached on the seven solder pads referred to in the article. Those seven pads include all logic signals and power required for the multiplexer. I installed wire-wrap pins in those pads, made a small board for the multiplexer from .100-inch hole-grid breadboard material, and pushed the small board over the wire-wrap pins, wire-wrapping the connections atop it to hold it in place. I found this less tricky than the piggyback soldering job would have been, and no leads of the multiplexer needed to be bent.

Fourthly, the multiplexer recommended is 74F253 or 74AS253; while these will do quite well, so will the 74F153 or 74AS153; which are slightly less expensive; the only difference is the nature of the DISABLE function, which (in this application) is always unused. All pin connections are identical to the '253.

I found that the multiple-layer board, having extra copper in the interior layers, was rather more difficult than most to desolder; I had to apply more heat (use a higher temperature on the desoldering iron) than usual, to complete the solder removal within a safe time. I did not notice any fragility of the printed circuit (no more than most, at any rate), and my procedure was, as I have mentioned, rather stressful. At one point, I fumbled, and dropped the board onto the concrete floor—there was no damage.

Prices were better than I expected, about $220 total cost for the chips, and that figure is dropping almost daily. Time required was almost exactly four hours. Since Apple charges about six hundred dollars more for the 512K Mac, it hardly pays to buy one; the fine folks in Cupertino are willing, in effect, to pay me handsomely for those hours, if I convert my 128K instead.

Instead of 'Macintosh' and 'Fat Mac', I will be using the terms 'Macintosh' and 'Anorexic Mac' in the future.

Sincerely grateful,
John Whitmore
FM-15 Physics Dept.,
University of Washington
Seattle, WA 98195

Going PUBlic

Dear *DDJ*:
PUBlic files have been used successfully on a wide variety of CP/M computers since the publication of our November 1984 [*DDJ*, #97] article "CP/M 2.2 goes PUBlic." However, we want again to caution readers that PUBPATCH must be installed on an *unmodified* CP/M 2.2 BDOS!

Several users have discovered that their Heath Co. CP/M 2.2.03 has been altered with a patch at 06E1 and 0DEE–0DF3 (relative to the start of the BDOS). The Heath patch causes the BDOS to stop building the disk group-allocation vector when it encounters the first directory entry having deleted data (E5) codes in both bytes 0 and 1. If PUBPATCH is then installed in this BDOS, the symptoms will be files that disappear from the end of the disk directory!

Heath users with this problem should patch location 06E2h back to its original value '06D2h' (relative) before installing PUBPATCH.

RELATIVE ADDRESS	06E1'
ORIGINAL 2.2 BDOS	JZ 06D2'
HEATH 2.2.03 BDOS	JZ 0EEE'

Two readers have reported problems with PUBLIC.ASM, used to set or reset the PUBlic file attribute bit. Updated code can be found in [Listing One (page 13)]. The first fix corrects a bug in v. 1.0 that prevents PUBLIC from setting the attribute for a very large file. The second takes care of a bug in the standard CCP (but not in ZCPR), which fails to restore the default dma address before continuing to execute a SUBMIT file.

No changes are required to PUBPATCH itself.

Sincerely,
Bridger Mitchell, Derek McKay
Plu*Perfect Systems
Box 1494
Idyllwild, CA 92349

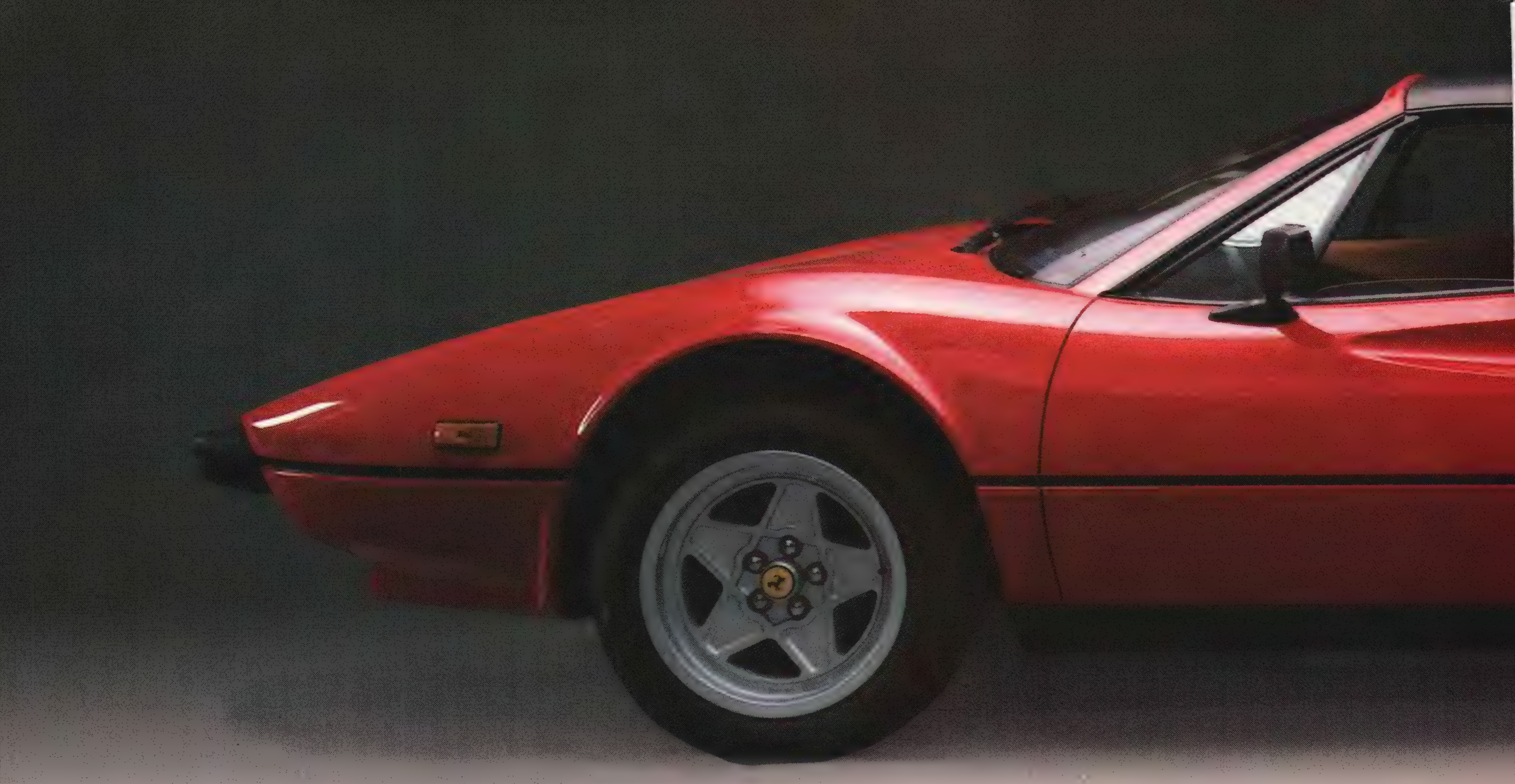
Turbo Bug

Dear *DDJ*:

I have been using Borland International's excellent Turbo Pascal compiler for several months now (I started with version 1) and I think it is by far the best implementation of any language I have every seen. The quick compiler and built-in editor have reduced the pain of correcting typos far enough to let me appreciate the virtues of Pascal. In the process I have discovered two things about the language that may be of interest to others. I am including listings of three short demo programs.

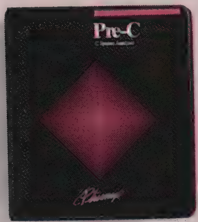
First, the "initialized constants" mentioned briefly in both the reference manual and the Turbo Tutor manual are really initialized static variables. This is clear from a close reading of either manual, but it is never stated in so many words, nor do any of the example programs demonstrate it. Compiling and running the [program in Listing Two (page 13)] will demonstrate that the typed constant I in the procedure ShowStatic behaves exactly like a static variable in C.

Compiling and running the program [in Listing Three (page 14)] will demonstrate a problem I discovered while trying to process the files produced by a linker cross reference utility. The Trunc and Round functions in both the MSDOS and CP/M versions of Turbo Pascal produce a run time



Programmers' Pfantas

Phoenix makes programmers' dreams come true. With the best-engineered, highest performance programming tools you can find. A full line of MS™-DOS/PC DOS programs and utilities no other company offers. All designed to help you write, test and deliver the best programs possible. Top-of-the-line quality at a price you can afford.



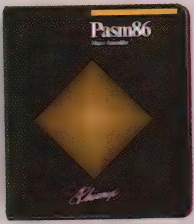
Finally, A Lint For MS-DOS.

Now you can get the full range of features C programmers working in UNIX™ have come to expect from their Lint program analyzer.

With Pre-C™ you can detect structural errors in C programs five times faster than you can with a debugger. Find usage errors almost impossible to detect with a compiler. Cross-check multiple source files and parameters passed to functions. Uncover interface bugs that are difficult to isolate. All in a single pass. Capabilities no C compiler, with or without program analyzing utilities, can offer. In fact, Pre-C outlints Lint, since you can handle analyses incrementally.

And, Pre-C's flexible library approach lets you maintain continuity across all the programs in your shop, whether you use Pre-C's pre-built libraries, pre-existing functions you already have, or some you might want to buy yourself.

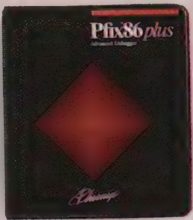
Plus, you're not limited to one particular library, and Pre-C keeps track of all the libraries you're using to make sure that code correctly calls them. **$395.**



Assemble Programs Twice As Fast.

Pasm™86 assembles Masm files 2 to 3 times faster than Masm 3.0. Pasm86 supports 8086/88, 8087, 80186 and 80286 processors.

With Pasm86's built-in defaults, you can write code quickly since you won't spend hours learning all the control statements needed at the beginning of your program. You can define symbols on the command line. Decide whether you want error messages or not. And, put local symbols within procedures. **$295.**



Still Fixing Bugs The Hard Way?

Pfix™86 Plus, the most advanced symbolic debugger on the market, eliminates the endless error searches through piles of listings. Locate instructions and data by symbolic name, using symbolic addresses. Handle larger, overlayed programs with ease.

An adjustable multiple-window display shows source and object code and data, breakpoint settings, current machine register and stack contents simultaneously. An in-line assembler allows program corrections directly in assembly language. Powerful breakpoint features run a program full speed until a loop has been performed n times.

With a single keystroke you can trace an instruction and the action will be immediately reflected in source, object, data, stack, and register windows. Another key begins a special trace mode that executes call and loop instructions at full speed. Designed to work with both Plink86 and MS™ LINK linkage editors. **$395.**

Special Thanks to Gasten Audrey of Framingham, MA.

ies by Phoenix.

Get The Lead Out Of Binary File Transfer.

Ptel™ is the universal binary file transfer program for MS-DOS 2.0 or 3.0. You can move binary files fast and accurately. Upload or download groups of files from Bulletin Boards or remote computers. Move files between dissimilar machines and operating systems.

Ptel's advanced binary protocol, Telink, offers better-than-Modem7 accuracy and performance. Faster transfer speeds. An on-screen update of error correction, blocks, transferred, and time to complete.

Includes popular Modem7 and XModem protocols. With Checksum or CRC. Plus Kermit and ASCII. **$195.**

Maximize Your Program's Efficiency.

Pfinish™ delivers the fastest running programs possible. This performance analyzer lets you ''zoom in'' on the inefficient parts of your program. Whether written in assembly language. C. Pascal. Fortran. Even Basic. Unlike profilers available today, Pfinish understands the structure of your program and reports the amount of activity and time spent in its subroutines or functional groups. Pfinish analyzes both overlaid and memory resident programs. Down to the instruction level. Reports are displayed. Stored on disk. Or printed out. In tabular form or histograms.

Do a dynamic program scan. Identify the most frequently executed subroutines. Find inefficient code that costs your program valuable time. Rank subroutines by execution frequency. **$395.**

Why Work With A Primitive Editor?

More than a powerful editor, Pmate™ is a text processing language. An emulator of other editors. A language-specific editor for C, Pascal, and Fortran. Pmate™ can even run in the background!

You get full-screen, single-key editing. Ten editing buffers. Horizontal and vertical scrolling. A ''garbage stack'' buffer. A built-in macro language with variables, control statements, radix conversion, tracing and 120 commands that you can group and execute with a single keystroke. **$225.**

Why Squeeze Your Program More Than You Have To?

The Plink™86 overlay linkage editor brings modular programming to 8086/88-based micros. Write large and complex programs without worrying about memory constraints. Work on modules individually, link them into executable files. Use the same module in different programs. Changes the overlay structure of an existing program without recompiling. Use one overlay to access code and data in other overlays.

Plink86 links Intel-format modules. **$395.**

Call (1) 800-344-7200. In Massachusetts (617) 762-5030. Or, write.

Phoenix

Phoenix Computer Products Corp.
1420 Providence Highway Suite 115
Norwood, MA 02062

Circle **no. 122** on reader service card.

error if you attempt to convert a real value of −32768 (8000H). Since this is a legal integer value, and right in the middle of the range of hex numbers, I think this has to be called a bug.

Listing Four (page 14) shows one way of programming around this problem. It works just fine when using real numbers and integers to represent hex numbers between 0 and FFFFH, but it would not be very useful in an application that requires both positive and negative real numbers.

Sincerely,
William Ames
8720 Topanga Cyn. Blvd. #8
Canoga Park, CA 91304

Borland assures us that this bug will be fixed in Version 3.0 of Turbo Pascal.—Ed.

Fast Hartley Transform

Dear *DDJ*,

We were pleased to see Ron Ullmans's letter in the December 1984, issue of *Dr. Dobb's Journal* regarding the Hartley Transform and the accompanying listing of a software program. We, too, consider its advantage to be great.

Your readers may wish to know Stanford University has proprietary rights on this technology, developed by Professor Ronald N. Bracewell. Patent rights are pending.

Use of the software without a license may be an infringement of those proprietary rights. However, a license under reasonable terms to the technology is available from Stanford.

Incidentally, we have called this development the "Bracewell" transform, in recognition of Professor Bracewell's fundamental contribution.

Sincerely,
Lisa Kuuttila
Consulting Associate
Office of Technology
Licensing
Stanford University

DDJ

Letters (Text begins on page 8)

Listing One

```
; UPDATE PATCHES FOR PUBLIC.ASM

vers    equ     1$2             ;update PUBLIC.ASM version number

; fix 1.  Insert the following code in the 'SAMEXT' routine
;         immediately preceeding its 'ret' instruction.

        rnz             ; v 1.1
        inx     h       ; extent is 0, check overflow (s2) ext.
        inx     h
        mov     a,m
        ani     7Fh
;
        ret             ; end of SAMEXT routine


; fix 2.  Insert the following macro statement at the label 'xit:'
;

xit:    dobdos  dmafn,80h       ;restore default dma for SUBMIT under ccp.
                                ; v 1.2
        lspd    ustack
        ret
```

End Listing One

Listing Two

```
TURBO PASCAL Program Lister, Copyright 1983 Borland International
Listing of: STATIC.PAS


program TestStaticVariables;
{ This program demonstrates that the "typed constants" of Turbo Pascal }
{ are really static variables. Each time the main loop calls ShowStatic }
{ the typed constant I is incremented by 1. When ShowStatic is called   }
{ again the incremented value is written to the CRT. }
var
  Ch  : Char;
  I   : integer;

procedure ShowStatic;
{ This prints and then increments the "constant" I }
const
  I : integer = 0;

begin
  Writeln(I);
  I := I + 1;
  end;

function UseHeap(L : integer) : integer;
{ This is included to demonstrate that the typed constant above is not }
{ distroyed when another procedure uses the heap or stack. }
var
  I : integer;
  J : ^integer;

begin
  I := -L;
  new(J);
  J^ :=  L;
  L := L + 1;
  if L < 5 then
    L := UseHeap(L);
  Dispose(J);
  UseHeap := L
  end;
```

(Continued on next page)

Listing Two

```
begin
  repeat
    read(Kbd, Ch);
    I := UseHeap(0);
    ShowStatic;
    Until Ch = #13;
  end.
```

End Listing Two

Listing Three

```
TURBO PASCAL Program Lister, Copyright 1983 Borland International
Listing of: CNVRT1.PAS


program RealToInteger1;
{ This program can be used to demonstrate a problem in }
{ converting real numbers to integers. }
var
  Nmbr : real;
  I    : integer;

begin
  Readln(Nmbr);
   { if Nmbr = 32768.0 the following produces a runtime error }
    if (Nmbr > 65535.0) or (Nmbr < 0) then
    { I am only interested in the range of 4 digit Hex numbers }
      Writeln(' overflow error')
    else begin
      if Nmbr < 32768.0 then
        I := Trunc(Nmbr)
      else
        I := Trunc(Nmbr - 65536.0);
      Writeln(I);
      end;
    end.
```

End Listing Three

Listing Four

```
TURBO PASCAL Program Lister, Copyright 1983 Borland International
Listing of: CNVRT2.PAS

program RealToInteger2;
{ this shows one way around the bug demostrated in version 1 }
var
  Nmbr : real;
  I    : integer;

begin
  Readln(Nmbr);
   { 32768.0 is converted correctly by the following code }
    if (Nmbr > 65535.0) or (Nmbr < 0) then
    { I am only interested in the range of 4 digit Hex numbers }
      Writeln(' overflow error')
    else begin
      if Nmbr < 32768.0 then
        I := Trunc(Nmbr)
      else
        begin
          I := Trunc(Nmbr - 65535.0);
          I := I - 1;
        end;
      Writeln(I);
      end;
    end.
```

End Listing Four

The Macintosh Software Library just got fatter . . .

It's about TIME. AND SPEED.

FASTER COMPUTER OPERATION with disk operations 2 to 5 times faster.

QUICK PROGRAM TRANSFER FROM OTHER CP/M COMPUTERS using built-in transfer utilities and standard CP/M file structure.

RUNS BIGGER MORE POWERFUL PROGRAMS with 52K transient program area available on 128K Mac, 300K available on 512K system and the capability is built-in to manage up to 16 Megabytes of RAM.

PLUG-IN WHAT YOU NEED ...CP/M for the Macintosh supports up to 2 printers and 16 Drives of 512,000,000 bytes each for a maximum of **8 Billion bytes of storage.**

MORE DISK SPACE AVAILABLE ... each disk drive has 360 bytes available, often eliminating the need for additional drives.

CHOOSE ANY PRINTER letter quality or dot matrix ... any serial printer will work just fine.

MEDIA PROBLEMS?...NONE ... all that is needed to transfer programs from 5¼ to 3½ is a modem transfer program to operate with our Modem 7 Communications Compatible Program ... this enables programs to be transferred from **XEROX, OSBORNE, HEATHKIT, KAYPRO, RADIO SHACK** and others.

THE 10 FEATURES:

1. 6 Disk System
2. Documentation from Digital Research and I.Q.
3. MacroAssembler ***(no extra charge)***
4. Modem 7 Compatible Communications Transfer Program ***(no extra charge)***
5. C Compiler ***(no extra charge)***
6. Text Editor ***(no extra charge)***
7. Menu Program (puts menus on your programs) ***(no extra charge)***
8. Standard Printer Driver ***(no extra charge)***
9. Copy Program ***(no extra charge)***
10. Leir-Seigler ADM3A Terminal Emulation Program ***(no extra charge)***

All This For Only $395 Retail

UNLIMITED EXPANDABILITY is made possible with the total and easy control of serial expansion ports which allows many forms of plug-ins.

UTILITIES ... such as STAT, PIP, DDT, ED, LINK68, INIT, NM68, SIZE, LO68, RELOC, and AS68.

AND We are currently compiling a vast Library of CP/M software that can be ported over to the Macintosh and our Macrolibrary will also be available soon.

NEED POPULAR WORD PROCESSING PROGRAMS? . . . Run WORDSTAR® DBASEII® AND other CP/M (8080) 2.2 Programs with our EM80 Emulator . . **ONLY $195.!**

With CP/M FOR THE MACINTOSH the Macintosh Software Library just got fatter!

Call or write for your Macintosh CP/M Software Catalog.

I.Q. software

I.Q. Software
2229 E. Loop 820 N.
Ft. Worth, Texas 76118
(817) 589-2000

DR. DOBB'S CLINIC

by D. E. Cortesi

Welcome, *Microans*

It seems that *DDJ* has picked up a batch of new readers, the erstwhile subscribers to the late, lamented *Micro*. Let's recap the purpose of this column for their benefit. It's supposed to be "a place for the display of techniques and discoveries." That's what we said the first time it appeared, in May 1981. Also, "We'd like to tell of unobvious uses for standard utilities. We want to uncover errors in documentation, to warn people away from pitfalls, and to show off those '*Eureka!*' moments that make systems work rewarding."

For a long time, most of the contributions to the Clinic involved CP/M and the Z80. Nowadays, we hear more from MSDOS/8086 users. Well, stand back, Zilog lovers; make room, Intel-lects! Now we have among us people who understand *real* computers like the 6502, the 6809, and the 68000 and *real* operating systems like OS9, SOS, and PRODOS. We can barely wait for the neat contributions they'll be sending in. Soon.

Wood Blocks

As bait for the *Micro* readers (hey, folks, we really like reader contributions here), we present our wood blocks program for the Atari. It's a little thing (see Listing at right) that we ran up one day while experimenting with the start, select, and option keys. These three keys control the three lowest bits of a byte in the Atari address space. The program samples that byte, and each time the combination of operated keys changes, it emits a heavily damped *klonk* at a different pitch. Play it by mashing down all three keys and then sort of working your fingers around.

The simulation of damped tones is pretty good. Experiment with the damping factor and with the expression that generates a note value from the key combination.

Roll Over, Keyboard

We've an interesting letter here from Charles Davis, who lives in a town with the delightful name of Flower Mound, Texas. Besides submitting some throughput timings for an Atari system (at last!), he's been studying keyboards:

"Perhaps you could survey another area of concern. I am either a very good typist or a very sloppy one! I need a keyboard with what used to be called 'N-key rollover.' These are hard to find. The current generation of computers and terminals does not seem to supply this feature. Most marketing and sales people . . . are unaware of the feature and therefore the sales literature does not mention it, even when it is present.

"A survey of the popular computers and terminals with reference to rollover would be interesting to me and, I think, to many of your readers. My procedure for checking rollover is to depress several keys in sequence, holding each until all are depressed, then I lift them all in the same order. I observe what happens on the screen. There seem to be three types of response:

- the first N keys (type 1)
- the last N keys (type 2)
- only the first and last keys (type 3)

The major variable is how big N is. Rarely must I find a helper with more fingers. One terminal manufacturer admitted that their terminals had N-key rollover and that N was supposed to be 5, but some key sequences did not work right so they would only guarantee N to be 3. This points up the weakness of my procedure: I do not do an exhaustive check and some problem characters may be missed. Without knowledge of the key matrix and the scanning algorithm, it would take too long to find the problems."

Davis sent along the results from some terminals he'd tested (see Table 1 on page 17). However, our first at-

```
10 REM WOODBLOX FOR ATARI, D.E. CORTESI
20 KEYBYTE=53279
   : DAMPING=0.29
30 SAMPLE=PEEK(KEYBYTE)
   : IF SAMPLE=BUTTONS THEN GOTO 30
40 BUTTONS=SAMPLE
   : PITCH=12*(8-SAMPLE)
   : VOLUME=15
50 SOUND 0,PITCH,12,VOLUME
   : VOLUME= VOLUME-VOLUME*DAMPING
   : IF VOLUME >= 1 THEN GOTO 50
60 SOUND 0,0,0,0
   : GOTO 30
```

Terminal	Response	Rollover
ADDS Regent 20	type 1	N > 10
ADDS Viewpoint	type 1	N > 10
Appollo DN600 Workstation	type 1	N > 10
Atari computers	type 3	n.a.
CIT 101E	type 1	N = 2
Daisy Logician Workstation	type 1	N > 10
Hazeltine 1410	type 3	n.a.
Heath/Zenith 19 and 29	type 4	N = 2
HP-85	type 3	n.a.
HP-9816	type 1	N > 10
IBM PC/XT	type 1	N > 10
Perkin-Elmer Bantam 550	type 1	N = 3
Soroc IQ-120	type 3	n.a.
Televideo 950	type 1	N = 2
Wyse 50	type 1	N = 3

Table 1
Charles Davis' survey of the number of keys that can be "rolled over" on different terminals. Types are: 1, first N keys; 2, last N keys; 3, first and last only; 4, first N keys then garbage.

Command Code	Type of Driver	Operation
0	either	initialize
1	block	media (sic) check
2	block	build Bios Parameter Block
3	either	IOCTL input call
4	either	input
5	character	nondestructive input no wait
6	character	input status check
7	character	input flush
8	either	output
9	either	output with verify
10	character	output status
11	character	output flush
12	either	IOCTL output call
13	either	notification of file-open
14	either	notification of file-close
15	either	IOCTL query: removable disk?

Table 3
The command codes to which an MSDOS device driver must respond, omitted from IBM's DOS *Technical Reference* manuals. Codes 13, 14 and 15 are new with PCDOS 3.0.–Codes 3, 12 and 15 are passed only to drivers that indicate IOCTL support in their Device Header Attribute word.

tempt to test a keyboard produced a new result that you should watch out for:

- the first N keys plus a garbage key (type 4)

This is the H19 problem we reported here in August 1982; it results from a lack of blocking diodes in the key matrix. Heath hasn't learned anything since, either. On our Heath/Zenith 29, holding L, I, and then S produces "LIQS" on the screen. We have to be very precise typing the word "list" or it comes out "liqst." Worse, some combinations produce false function keys. The combination E, I, then P generates a false down-arrow (escape-B) sequence, for instance. So the H/Z29 is an N=2 machine, even though for some key combinations it will pass Davis' test as a type 1 with much higher N. Incidentally, the good combinations occur on the home row. That suggests that you might apply Davis' test on the top or bottom rows.

Charting DOS

We've been boning up on MSDOS and PCDOS lately, in preparation for writing a book and we've compiled the chart of DOS services that appears in Table 2 (page 19) as a study aid. For each possible Int 21 service it shows the request number in decimal and hex, the releases that support it, and a capsule description of the service. The services are organized into groups by function. The groups and the services within the groups are generally in ascending order by request number, but functional relationships are given precedence.

The idea is that you can look for the service you need by scanning a small group of related ones. The capsule descriptions are just enough to differentiate between similar functions. Having found the service you need, you can access its detailed description in your DOS manual through its request number.

Our study of PCDOS has been impeded no small amount by the poor quality of the IBM DOS manuals, especially those for DOS 3.0. For one example, the documentation on service 56/38h (get or set country-dependent information) is so badly done that we honestly cannot make sense of it. If you are a reader who can write a comprehensible, accurate description of the DOS 3 version of service 38h, please do so and send it to the Clinic.

For another example, the chapter on device drivers omits any specification of the command codes to which a device driver is supposed to respond. In the DOS 2.1 manual you could still cull the essential information from the comments of a sample program. In DOS 3.0, the sample program has been dropped from the manual. The book says just enough that you can infer that command codes 3, 4, 8, 9, and 12 specify input or output, but there is no way you can tell which code calls for what action!

Fortunately, the missing information can be found in Microsoft's version of the technical manual, from which we constructed Table 3 (page 17). Although some command codes are specific to character or block devices, it seems wisest for all drivers to expect all command codes. They can treat the unwanted ones as null operations, or they can reject them with the "unknown command" error code.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 190.**

Service number in decimal and hex	Appears in which releases	Functional group	Capsule description
0/00	1/2/3	program cntrl	end program (requires PSP *CS)
49/31	–/2/3	program cntrl	end, stay resident, AL=ret, DX=size
76/4C	–/2/3	program cntrl	end program, AL=ret
38/26	1/2/3	program cntrl	make daughter PSP at (DX:00)
75/4B	–/2/3	program cntrl	(AL=0) load, run daughter from path *DS:DX
77/4D	–/2/3	program cntrl	get AH=term code, AL=ret of daughter
75/4B	–/2/3	program cntrl	(AL=3) load DS:DX —>path as overlay
98/62	–/–/3	program cntrl	get BX=paraddr of my PSP
48/30	–/2/3	system info	get DOS version number
42/2A	1/+/3	system info	get CX=year, DH=mo, DL=day (2,3:AL=dow)
44/2C	1/2/3	system info	get time as CH:CL:DH.DL
51/33	–/2/3	system info	(AL=00) get Break switch in DL
53/35	–/2/3	system info	get int(AL) vector to ES+BX
56/38	–/2/+	system info	(AL=00) get country-dependent info
72/48	–/2/3	system info	(BX=FFFF) get size available RAM to BX
89/59	–/–/3	system info	get extended error to AX, BH, BL, CH
37/25	1/2/3	system control	set int(AL) vector to (DS+DX)
43/2B	1/2/3	system control	set date from CX, DX
45/2D	1/2/3	system control	set time from CH:CL:DH.DL
51/33	–/2/3	system control	(AL=01) set Break switch from DL.0
56/38	–/–/3	system control	(??) set country-dependent info
72/48	–/2/3	system control	allocate BX paragraphs
73/49	–/2/3	system control	return RAM from ES=paraddr
74/4A	–/2/3	system control	resize ES=paraddr to BX paras
1/01	1/2/3	kbd/handle 0	get key: wait, echo, break-check
6/06	1/2/3	kbd/handle 0	(DL=FF) get key: noecho, nowait, nobreak
7/07	1/2/3	kbd/handle 0	get key: wait, noecho, nobreak
8/08	1/2/3	kbd/handle 0	get key: wait, noecho, break
10/0A	1/2/3	kbd/handle 0	buffered line input with editing
11/0B	1/2/3	kbd/handle 0	test if key available (break)
12/0C	1/2/3	kbd/handle 0	flush buffered keys, then do (AL)
2/02	1/2/3	scrn/handle 1	put DL with editing and break-check
6/06	1/2/3	scrn/handle 1	(DL < > FF) put DL, noedit, nobreak
9/09	1/2/3	scrn/handle 1	iterate DOA 2 over *DS:DX to $
3/03	1/2/3	aux/handle 3	get byte from COM1
4/04	1/2/3	aux/handle 3	put byte to COM1
5/05	1/2/3	prt/handle 4	put byte to LPT1
25/19	1/2/3	disk info	get AL=default drive
47/2F	–/2/3	disk info	get Disk Transfer Address to ES:BX
27/1B	1/+/3	disk info	get FAT *DS:BX (2,3: only 1st byte of FAT) . .
28/1C	–/2/3	disk info	. . and CX=bytes/sec AL=sec/unit DX=units/disk
54/36	–/2/3	disk info	BX=free units, CX/DX/AL as for 1C
84/54	–/2/3	disk info	get write-verify switch to AL.0

Table 2
Chart of DOS Services

(Continued on next page)

(*Continued*)

13/0D	1/2/3	disk control	reset disk system
14/0E	1/2/3	disk control	select drive, return count of drives
26/1A	1/2/3	disk control	set Disk Transfer Address = DS:DX
46/2E	−/2/3	disk control	set write-verify switch from AL.0
15/0F	1/2/3	fcb control	open file from FCB
16/10	1/2/3	fcb control	close file from FCB
17/11	1/2/3	fcb control	scan current directory for first match
18/12	1/2/3	fcb control	scan current directory for next match
19/13	1/2/3	fcb control	erase files matching FCB
22/16	1/2/3	fcb control	make new file from FCB
23/17	1/2/3	fcb control	rename file from FCB
35/23	1/2/3	fcb control	get size of file to fcb
36/24	1/2/3	fcb control	set relrecno from current file position
40/28	1/2/3	fcb control	(CX = 0) trunc or stretch file to size
41/29	1/+/3	fcb control	parse filespec *DS:SI into fcb *ES:DI
20/14	1/2/3	fcb i/o	read next record to DTA
21/15	1/2/3	fcb i/o	write DTA to next record
33/21	1/2/3	fcb i/o	read record at direct address
34/22	1/2/3	fcb i/o	write record to direct address
39/27	1/2/3	fcb i/o	block read CX records to DTA
40/28	1/2/3	fcb i/o	(CX > 0) block write CX records from DTA
57/39	−/2/3	directories	make directory for path *DS:DX
58/3A	−/2/3	directories	erase directory at end of path *DS:DX
59/3B	−/2/3	directories	set current path as path *DS:DX
71/47	−/2/3	directories	get current path of DL=drv to DS:DI
63/3F	−/2/3	file i/o	read CX bytes from BX=handle to DS:DX
64/40	−/2/3	file i/o	write CX bytes to BX=handle from DS:DX
60/3C	−/2/3	file control	create/trunc file (path *DS:DX, CX=attr)
91/5B	−/−/3	file control	create new or fail (path *DS:DX, CX=attr)
90/5A	−/−/3	file control	create unique file (path *DS:DX, CX=attr)
61/3D	−/2/+	file control	open file (path *DS:DX, AL= access type)
62/3E	−/2/3	file control	close file (BX=handle)
65/41	−/2/3	file control	erase file (path *DS:DX)
66/42	−/2/3	file control	seek (BX=handle, AL=method, CX+DX=offset)
67/43	−/2/3	file control	(AL=0) get CX=attr of file path *DS:DX
67/43	−/2/3	file control	(AL=1) set attr of file path *DS:DX to CX
69/45	−/2/3	file control	create a duplicate file handle
70/46	−/2/3	file control	force a handle to be a duplicate
78/4E	−/2/3	file control	search 1st match to path *DS:DX
79/4F	−/2/3	file control	search next match to path *DS:DX
86/56	−/2/3	file control	rename/move path *DS:DX to path *ES:DI
87/57	−/2/3	file control	(AL=0) get CX=time, DX=date of path *DS:DX
87/57	−/2/3	file control	(AL=1) set CX=time, DX=date of path *DS:DX
92/5C	−/−/3	file control	(AL=0) at CX+DX in BX=hdl, lock SI+DI bytes
92/5C	−/−/3	file control	(AL=1) release bytes locked by prior call

Table 2
Chart of DOS Services

REALIZABLE FANTASIES

Dr. Dobb is a Subversive

by D. E. Cortesi

The publishers of DDJ *recently threw a party to celebrate the 100th issue of the magazine. It was a friendly affair, with lots of sincere speeches about how important* DDJ *has been and about how it still has a great role to play. There was a lot of talk about "hackers:" what they accomplished; where* DDJ *stood in relation to them; where they might have gone now.*
they were expressed, the sentiments seemed incomplete to me. Something was being overlooked, but I couldn't figure out what it was until I was halfway home. Then I lay awake working out the speech that ought to have been given, but wasn't. And this is it . . . .

We've heard a lot about "hackers" in the last few minutes. I would like to protest the use of that word. I understand why it's being used. It's because nobody can think of the right word to describe the people who founded *DDJ* and the people who read and profited from it in its earliest days. We know they weren't part of the establishment; we know they didn't play a conventional role in the society of their day. We have, perhaps, an uneasy feeling that we aren't quite like them and that they wouldn't find us very interesting—even though perhaps we *were* them just a few years ago.

So we scratch around in our vocabularies, searching for the right word to describe these departed people, and the best we can come up with is "hacker." Well, it's the wrong word.

The right word is . . . *revolutionary. Subversive. Bomb-throwing anarchist.*

To me, the hacker state is a frame of mind in which one is turned inward on oneself and the machine: one's spiritual third eye rolls up into the head. The people who birthed *DDJ* and personal computing in general were quite the opposite. Their eyes were turned outward on society—and twinkled with mischief.

They were bent, quite consciously, on overthrowing the established order. It was to be a nonviolent overthrow, one conducted if possible with whimsy and good humor, but it was to be an overthrow, a toppling, nonetheless.

The central tenet of this jolly revolution was (and still is) the belief that *if we put enough computers in the hands of enough people, wonderful changes will take place.* What the wonderful changes are . . . well, that varies with the year and the person you're talking to. But that they will be wonderful, as well as radical in the deepest and best sense of the word, no computer subversive doubts. I certainly don't.

The founder of the movement and its only theorist, its Marx or Che Guevara if you like, is Ted Nelson. His book, *Computer Lib*, was written in reaction to computers, their employers, and their professional priesthood as those things existed in the early 70s. It was a protest: "Either computer systems are going to go on inconveniencing our lives, or they are going to be turned around to make life better," and then, in caps, a battle cry: "COMPUTER POWER TO THE PEOPLE! DOWN WITH CYBERCRUD!"

And to make sure we didn't miss the point, there was the symbol of the clenched fist on the cover. I hope I'm not the only one here who remembers when the word *Lib* and the upraised fist were potent, dangerous symbols. That Nelson borrowed both for his manifesto indicates how serious he was.

I don't know if the People's Computer Company admitted to subversive intentions in public or not. Unfortunately for me, I was out of the country during its glory days. I was aware of micros and an early subscriber to *DDJ*, but I was only a fellow traveler to the revolution, not a revolutionary myself.

But whether the intentions were admitted or not, I'm positive that they were there. No one who's spent any time with Bob Albrecht, or with Ramon Zamorra, who founded ComputerTown USA!, could have any doubt that these people are motivated by a deep love of mischief. Nothing pleases them better than a well-toppled apple cart or a well-rocked boat.

Another revolution that I missed out on was the birth of the acid culture in the early 60s. But I suspect that the spirit of People's Computer Company had a lot in common with the spirit of Ken Kesey's Merry Pranksters, and with that of the Diggers and the various communards of the 60s. They all felt that they'd stolen the fire of the gods and that they'd damn well better set as many blazes as they could before the gods came to take it back.

After all, what could possibly be more subversive than training children to operate the tools of the establishment, *without* at the same time inculcating the establishment's line of thinking? But that was what Albrecht, Allison, *et al.*, were doing. I don't know if this is what was specifically in their minds or not. But consider. You have all had the following experience. You program something in BASIC; you find that the program doesn't work because of the inevitable bugs; then, after you pick them off, you find that it *still* doesn't work because the limited accuracy of binary floating point has turned your nu-

merical results into oatmeal. Nobody who has been through that can ever look at their bank statement in quite the same way again. Or their phone bill. Or, if there is a live connection between one side of their brain and the other, can they ever feel quite the same about a proposal for an antiballistic missile system as they might once have done before their personal encounter with computers in the raw.

Albrecht and company used timesharing terminals to start indoctrinating children and susceptible adults, but then the micro came along. The earliest micros were pitiful, but to the eyes of a computer subversive they appeared to be the sword of liberation. A computer you could hold in your hand!

For a parallel, try to imagine that Gutenberg had never invented his printing press; that the only way for books to be reproduced was for them to be handwritten by scribes. Of course, only the wealthy and the government can afford to pay scribes and buy vellum, so the only thoughts that circulate in book form are the ones acceptable to the powers that be. Then somebody invents the typewriter: a cheaper, faster way of being a scribe, one that anybody can learn. So you start a counterculture book factory, teaching people to use typewriters to duplicate unapproved literary works.

And then somebody walks in the door with the latest invention, *the Xerox copier*. And just that simply, your revolution has been won; now it's just a matter of mopping up.

In the case of the computer revolution, the mopping up is still going on. The micro *has*, in fact, hurt the computer bureaucrats, the people who would try to make you swallow an inconvenience on the grounds that the computer wouldn't let them be helpful. These days they find it a lot harder to say "the computer won't let me" because so many clients are likely to answer "Why not? My computer would." There really is a lot less these days of what Nelson called cybercrud, and we can credit that to the wide popular exposure to micros.

But there is a lot more to be done. We've spread computers all over the place and still have no millenium, nor anything remotely like it. There's been no great liberation of minds or spirits. The revolution has been assimilated, just as the automotive, sexual, drug, and free speech revolutions were assimilated before it, and the feminist revolution is being assimilated now: they come, they pass, things are different but not very much so. Surely by now the wonderful thing should have happened. Surely by now somebody should have written the program that will set us all free? But no.

One reason for this failure is that the computer did not turn out to be a sword that could be placed in anyone's hand. It isn't a sword at all, nor anything like a simple edged or pointed weapon. For complexity it's more like a whole battalion of soldiers. If you tell some poor shmuck to start wielding one, you put him in the position of a private promoted instantly to general and asked to take over management of a battle.

To change metaphors nimbly in midstream, we have made the belated discovery that the computer priesthood that we deposed was not entirely a group of fakes; they had more

than mumbo jumbo going for them. True, they'd made public access to the machines difficult, or allowed it to be difficult. But now that we have our own machines, we find out that there were good reasons for that. It turns out that quite possibly the most difficult thing you can do with a computer is to make it be truly helpful. If you are in a hurry for an answer, you will find it a lot easier to program the machine to take its input in rigid, unforgiving formats, and to give cryptic, insulting error messages, than to make it flexible and friendly and forgiving.

In fact, we find that in putting on its happy face, the machine uses up so much capacity that it has precious little left in which to do any useful work.

Partly this is a real effect. It truly is difficult to make a computer be helpful, and there are deep, unsolved problems in the way of making that happen. But in part, I think that the revolutionaries were a bit hasty in exiling those priests. They have a genuine science, computer science, that deals with the elegant and efficient application of machine power. All too little of that science is being applied today. The result is that we have machines with 512K of storage that can't keep up with the demands of a single user. Damn it, we used to support twenty APL users on a single 360 model 40 with 128K; we considered a 370/158 with 512K a big machine, suitable for use by a middle-sized company with half a dozen programmers online to TSO or VM/370. The *big* macro assembler for the IBM 370 needs a 128K partition to run in. That's the assembler program plus all the tables and work areas and buffers it uses. The macro assembler for the IBM PC can barely load *itself* into 128K, and it can't assemble anything of useful size until you give it 256K. In short, there has been a major loss of efficiency in moving from the software that the priests built for the corporate mainframes to the software that we are putting on personal computers today.

As a good revolutionary, I must believe that computer *use* is for everybody, and I really do. But programming the computers so they can *be used* is almost certainly a job for specialists.

The question that confronts us now is: whose specialists are they going to be? Will they be the tools of the new computing establishment, working for the fiscal advantage of a company like IBM or AT&T or Microsoft, or for the political advantage of a nation like Japan or France?

Or will they be grass-roots benefactors like so many who supplied the software we have now—and who often supplied it through the pages of *DDJ*? I would like it to be the latter. In fact, I propose to you that what *DDJ* has always done is to support and train and encourage our grass-roots systems programmers. And that is still the best, and most truly radical, thing that it can do in the future. I'm delighted to see that its editors agree with me. Proof that they do is in their publishing Richard Stallman's "GNU Manifesto" (*DDJ* #101, March 1985), an outrageous, revolutionary challenge quite in the old vein.

My own best writing for *DDJ* has

been inspired by a species of the Robin Hood ethic: a desire to steal ideas from the well-funded and put them into the heads of the unfunded. I enjoyed writing my recent article on Prolog for just that reason. Prolog and the ideas behind it had been the property of a few academics, and these ideas were rapidly being co-opted by the Japanese for their Fifth Generation. But *DDJ* gave me a chance to snatch those same notions and spread them around everywhere, making them accessible to everybody who can grasp them. Of course the real revolutionaries are the people in London who publish Micro-PROLOG and the ones in this country who are selling a version of DEC-10 Prolog for the IBM PC. The ideas wouldn't be much use without the cheap software that implements them, and that wouldn't be of any use without the cheap micros to run it on.

So these processes all feed back on themselves and get richer and deeper. But the continuation of this revolution, at least its continuation *as* a revolution and not as a captive toy of the corporations and the advertisers, depends on getting ever more sophisticated ideas and tools into the hands of creative people, so that they can keep putting ever more sophisticated programs back into ours.

Notice how the ante goes up on every round. The level of sophistication is escalated every time through the loop. The Mac's internal toolkit, and its competitive followers Windows and GEM, are vastly harder to use well than the old CP/M BDOS functions. Graphics, AI, and speech recognition, in their different ways, require the exercise of all the algorithmic tools of computer science. And it takes very abstruse, very mathematical tools to analyze these massive, sophisticated programs and prevent them from overflowing the capacities of personal hardware, which really are quite limited. The experience of mainframers has been that your hardware is *always* an order of magnitude too small for the software you'd like to run. There's no indication that this law has been repealed for micros. We will be squeezing quarts into pint bottles forever, and it takes sound theoretical insight to do that well.

I should add that the rule that the capacity of the hardware always falls short of the needs of the software just *might* be repealed by highly parallel, multiple-micro systems. Two things about such systems: first, they are right on the fringe of computer science and the academics haven't the foggiest idea how to build good software for them; and second, they are ideally suited for hobbyist-level, garage experiments.

But this is why the face and contents of *DDJ* have been changing. Whether the editors and contributors knew it or not, they've been trying to prepare themselves to discuss and understand computers at the more sophisticated level permitted by the second generation of micros. But there's been no change in who is doing the discussing, nor in their essentially subversive motive, which is to snatch the tools of the establishment and apply them in the public domain.

It's that mischievous, subversive motive that has kept me writing for *DDJ*. That, and the belief I still hold that, somewhere out there, fingers on the keyboard of an Apple or Osborne, is a fifteen-year-old kid who is capable of writing the program that will turn the world upside down. The proper role of *DDJ* is to make sure that that kid has free, unfettered access to the ideas and the software tools that she needs in order to write it.

DDJ

Reader Ballot

Vote for your favorite feature/article.
Circle Reader Service **No. 191**.

INTRODUCING THE MODULA-2 SOFTWARE DEVELOPMENT SYSTEM

The first complete programming environment brings the industry to an all-time low.

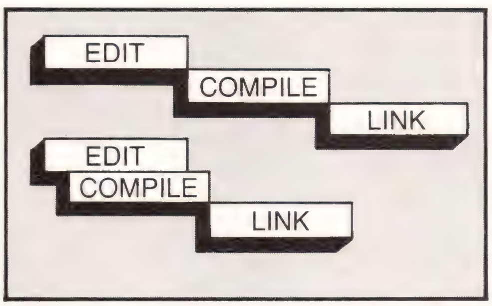
$80.88

Modula-2 has been hailed as the programming language of the future. Its modular design and built-in error-control features make programming more efficient than ever.

And now there's a system that makes programming more affordable than ever. Interface Technologies' Modula-2 Software Development System (M2SDS).

**EASY TO LEARN.
EFFICIENT TO USE.**
M2SDS features a "syntax-directed" editor that makes programming easy for beginners to learn. And faster for professionals to use.

With our editor you can enter full statements with a single keystroke. And save up to 90% on typing time. It also gives on-line help in correcting undefined variables and syntax errors—which saves even more time.



It not only has a faster compiler, it also saves time by compiling while you edit.

Multiple editor windows let you refer to one file while you edit another. That's one more way M2SDS adds hours of more creative, more productive time to your day.

Work faster and easier with multiple editor windows.

TURN "WAIT TIME" INTO "WORK TIME." When there's no time like real-time, you can count on the M2SDS compiler. Up to 100 lines of Modula-2 text can be turned into native machine code in less than five seconds.

To create programs using your computer's full capacity, there are 18 library modules. And unlike other low-priced compilers, M2SDS has a linker that assembles the components of your program. Automatically.

**BREAKTHROUGH TECHNOLOGY.
BREAKTHROUGH PRICE.** M2SDS works with IBM® PC, XT, AT or any other 100% compatible computer. Any programs you develop, you own. And M2SDS is non-copy protected.

For just $80.88, M2SDS is the complete programming environment. Including editor, compiler, linker, library modules, 8087 support and more.

Or choose the expanded, fully upgradeable SDS-XP for just $249. Later you can add a debugger, foreign object import module and tool box for even more programming capability. And efficiency.

So whether you're a professional looking for a faster way to program, or a novice looking for an easier way to learn, there's a Modula-2 Software Development System just for you.

Call us today for more information or to order your M2SDS. Find out how our new low in system pricing can put your programming efficiency at an all-time high.

WE ACCEPT CHECKS, MASTERCARD, VISA AND AMERICAN EXPRESS. Price does not include shipping and handling. Texas residents add 6.125% Sales Tax. International orders add $30.

INTERFACE TECHNOLOGIES
3336 Richmond, Suite 200, Houston, TX 77098

GET MORE PROGRAMMING EFFICIENCY IN A SYSTEM THAT COSTS LESS. IN TEXAS, CALL (713) 523-8422.

CALL 1-800-922-9049

Circle **no. 50** on reader service card.

C CHEST

Command Line Processing

by Allen Holub

A command line switch is an argument usually preceded by a '–' that modifies the way in which a program works. Just about every C program I write uses command line switches, and all these programs have to process these switches one way or another. For a long time, I wrote a different little subroutine for each program, modifying the routine to deal with the idiosyncrasies of a particular program. After writing the same subroutine about a million times (and making the same off-by-one error with argc every time), I thought that there *had* to be a better way. All my processing routines were essentially the same in structure; only the names of the command line switches were different. A general purpose command line processor was clearly what I needed.

Looking around in the literature for something that did the job, I found the "Argum" package, which appeared in Anthony Skjellum's C/Unix column in *DDJ* (#70, August 1982). The program was well written and did what I needed, but it had several problems. First of all, it did a lot more than I needed and the extra functions added extra code. Because this package was going to be included in every program I wrote, a reduction in scope seemed to be a good idea. Another, related, problem was the internal tables used by Argum. They were created at runtime (as compared to compile time). This added both extra code and extra execution time to any program that used Argum. Finally, Argum had no convenient way to deal with errors on the command line. In view of these problems, I decided to write my own processor, a description of which follows. This is by far the most frequently used subroutine in my standard library.

Getargs( )

The command line processor is actually a package of subroutines. Access to these routines is through a single procedure, called getargs( ). The routines are table driven. In every program you have to declare a table in which the various command line switches are described. Getargs( ) removes the switches from the command line as it works. This means that position-sensitive arguments (e.g., a switch that applies to all files that follow it on the command line, but not to any files that precede it) are not supported. I don't use this sort of switch very often, and I got tired of having to skip past arguments that had already been processed to get to a filename. The arguments are evaluated left to right, so interaction between PROC type switches (see below) is possible.

When getargs( ) finds an error on the command line, it prints to stderr a list of all legal switches, along with a brief description of what those switches do and their default values. After it prints the error message, getargs( ) terminates the program with an exit(1) call. This closes any open files and returns to the operating system.

Command Line Switch Formats

Command line switches all must take the form:

–<character> [<number> | <string>]

That is, all arguments start with a minus sign; each switch is identified by a single letter that follows the minus sign immediately (no intervening spaces); the letter may be followed by an optional number or string, again with no spaces between the character that serves as the switch identifier and the corresponding number or string. Switches may be combined, provided that the argument types allow this combination. For example, the command line:

program –a –b123 –c

can also be written as:

program –ab123c

However, if getargs( ) expects a string to follow the switch identifier character, then it assumes that the rest of the argument is part of the string. You have to be careful when combining string type command line switches with other types.

Using Getargs( )

To use getargs( ) you must do two things: (1) set up a table to tell the routine what kind of command line switches to expect; (2) call the routine itself somewhere early in the main( ) module. The file getargs.h (Listing One, page 32) contains the #defines and typedefs needed to create the command switch descriptor table. This table is an array of structures:

```
typedef struct
{
    unsigned   arg : 7      ;
    unsigned   type : 4     ;
    int        *variable    ;
    char       *errmsg      ;
}
ARG;
```

The arg field is a single character that identifies the switch on the command line. In the following descriptions this character is represented as <switch>. The type field may take any one of five values, all of which are

#defined in getargs.h. The behavior of getargs( ) will vary according to the value.

INTEGER switches take the form

−<switch> <number>

Numbers preceded by 0x are hex, by 0 without the x, octal. All others are decimal. The number is terminated by any character not legal in the indicated radix. Any characters that follow are assumed to be additional switches. The int sized variable pointed to by the variable field of the ARG structure will be set to the value of the <number>. If the <number> isn't typed on the command line, *variable will be set to 0.

BOOLEAN switches will cause some action based on the presence or absence of the indicated switch on the command line. If the switch is present, then the int pointed to by the variable field is set to 1, otherwise *variable is not modified.

CHARACTER switches take the form

−<switch> <character>

When the switch is found on the command line, then *variable is set to the character immediately following the <switch> character.

STRING switches take the form

−<switch> <string>

When the switch is found on the command line, the *character pointer* pointed to by the variable field is set to point at a string consisting of all characters following (but not including) the <switch> character up to the end of the current argument (not to the end of the command line). In the case of combined switches in a single argument, the STRING argument must be the last one because all following characters will be considered part of the <string>. When defining a STRING switch in the table, be sure to cast the variable into an integer pointer. See Listing Three (page 38), line 15, for an example.

PROC switches take the form

−<switch> <anything>

This works like the STRING switch in that all characters following <switch> up to the end of the current argument will be part of the <anything>. However, the variable field is a pointer to a subroutine that is called indirectly as soon as the switch is encountered on the command line. A pointer to <anything> is passed to this subroutine as a single argument. An example of such a subroutine is given in Listing Three, line 19. It is the responsibility of the called subroutine to parse <anything> as appropriate.

The errmsg field is used to print an error message if an undefined switch is found. An example of the format is shown in the Figure (page 30). This message was generated when the command line "argtest −x" was given to argtest.

Listing Three (if you haven't realized it by now) is an example of how to use getargs( ). Lines 5-9 are declarations of the objects that are used in the variable fields of the switch descriptor table. The initial values of

these variables remain unchanged if the switch is not encountered on the command line at runtime. The table itself, Argtab, is declared on lines 10-17. The main( ) routine calls getargs( ) on line 35. The rest of main( ) just prints the command line, both before and after the getargs( ) call, so you can see how getargs( ) strips processed switches out of argv.

Getargs( ) itself starts on line 98 of Listing Two (page 32). It is passed four arguments: argv, argc, a pointer to the switch descriptor table (tabp), and the size of this table in elements (not bytes). The routine returns a new value of argc, and argv is compressed, i.e., all entries containing command line switches are removed from it. The for loop on line 112 processes argv one element at a time. Nargv points into argv and strips processed switches. Nargv is initialized to point at argv[1] because argv[0] can't possibly contain a command line switch (it holds the program name). If no leading minus sign is found in the argument, *argv is copied to *nargv and both pointers are advanced. In addition, nargc (new argc) is incremented. If the argument does begin with a minus sign, it is processed as a command line switch. In this case *argv is not copied to *nargv, and argv, but not nargv, is advanced. This effectively eliminates the argument containing the switch from the argv array.

Findarg( ) (Listing Two, line 44) is used by getargs( ) to see if an argument is in the table. It performs a linear search. Because the table is usually fairly small, it seemed as if the extra code needed for a binary search wasn't justified. Findarg( ) takes three arguments: c is the switch identifier character being searched for; tabp and tabsize are the same parameters as were passed to getargs( ).

Setarg( ) (Listing Two, line 9) is called when a command line switch is found in the table. Argp is a pointer into the table, as returned from findarg( ). Linep is a pointer into the argv entry that is being processed. Casts were used to make this routine as transportable as possible.

Pr_usage( ) (Listing Two, line 57) is used to print the error message shown in the Figure when an illegal command line switch is found. It just spins through the table, printing the current contents of the object pointed to by the variable field as well as the message given in the errmsg field.

The final routine in the package is stoi( ) (for string to integer, Listing Four, page 38), which is used to process INTEGER switches. Stoi( ) is a fancy version of the standard library routine atoi( ) (described on page 58 of Kernighan & Ritchie). Stoi( ) can handle numbers in base 8, 10, or 16. If code size is a real issue, you may want to remove the code that processes octal numbers (Listing Four, lines 45-53). Another major difference between stoi( ) and atoi( ) is that stoi( ) is passed a *pointer* to a character pointer. That is, it is passed the *address* of the pointer, which in turn points into the string to be processed. This extra level of indirection lets you modify the character pointer itself to point past the end of the number being processed. If you use atoi( ), you have to go through the string twice, once to extract the number and once

Illegal argument <x>. Legal arguments are:

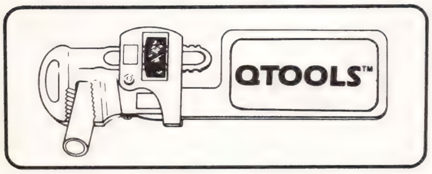
−b	boolean argument	(value is FALSE)
−c<c>	character argument	(value is .)
−n<num>	integer argument	(value is 0)
−s<str>	string argument	(value is <doo wha>)
−p<str>	procedure argument	

Figure
Error Output from Program in Listing Three

Let Us Mind Your Q's and T's

Quality Tools from QCAD Systems, Inc.

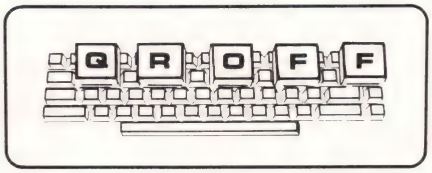
Tools



A UNIX-like toolset that runs under MS-DOS, making full use of DOS I/O redirection and environment variables. Great for pattern search, substitution, translation, file listings, and maintenance. On-line help and detailed user manual with extensive examples. Increases programmer productivity by providing a concise way to specify complex file manipulations. Some tools are versions of UNIX programs -- cat, diff, echo, fgrep, grep, head, ls, more, mv, pr, split, subst, tail, wc -- while others are new; 19 in all! *Runs on the IBM PC.*

$49.50

Typesetting



A typesetting system for the **IBM PC** and **HP LaserJet** printer. Ordinary text files, with some simple command strings added, can be printed quickly and easily with any of these features: mixed fonts, underlining, headers, footers, superscripts, subscripts, left & right margin alignment, page numbers, line spacing, indentation, automatic index generation, table formatting, spacing up or down, and more! Documents, letters, memos, and tables can be formatted and printed as nicely as a print shop could do them! (Requires HP font cartridge -- call for details.) *Runs on the IBM PC and HP 9816.*

$79.95

Translation

An LALR(1) parser generator that works from grammar, semantics, and skeleton files. Skeleton files provided for **C** and **Pascal**; easily adaptable to other languages. Multi-level interactive debugging support. Advanced error recovery mechanism. Sample translators included. Generates a complete parser and lexical analyzer in source code. Extensive documentation and examples. **Turbo Pascal** included. Educational discount available. Accompanying textbook (2nd ed.) available in late 1985. *Runs on the IBM PC, DEC Rainbow, and HP 9816.*

Full System $995.00
Demo System $ 25.00

WRITE OR CALL FOR MORE INFORMATION.
TOLL FREE LINE: 800-538-9787

In California, call 408-255-5574. California residents please add sales tax. Check, money order, Mastercard, and Visa accepted. [UNIX is a trademark of AT & T Bell Laboratories. IBM PC is a trademark of IBM Corp. Turbo Pascal is a trademark of Borland International. LaserJet is a trademark of Hewlett-Packard Co. QTOOLS, QROFF, and QPARSER are trademarks of QCAD Systems, Inc.]

QCAD
SYSTEMS, INC.
1164 Hyde Ave., San Jose, CA 95129

Circle **no. 51** on reader service card.

more to skip past the characters that represent that number.

Conclusion

Several additions could be made to getargs( ), at the cost of an increase in complexity and code size.

- A common command line function is to open or close a file in a particular mode and to abort if the file can't be opened. At present I'm doing this with a PROC type command line switch, but you could add another argument type, whose variable field points to a FILE pointer. Alternately, variable could point to a structure that included the FILE pointer, an open mode, a pointer to an error processing routine, and a default file name.
- Another nice feature would be to mark an argument if its presence is required on the command line. You could accomplish this by adding additional types (i.e., REQ _INTEGER), or by adding another field to the ARG structure. An error message would be printed if the argument wasn't found on the command line.
- Getargs( ) depends on initializers to set default argument values. If your compiler doesn't support initializers, you can add a default value field to the ARG structure.
- I have violated my smallness rule by using stoi( ) to process numeric arguments. If you expect only decimal numbers, you may want to replace stoi( ) with atoi( ). However, stoi( ) is a useful routine in its own right.

I'm sure that you can think of other bells and whistles. I've been using getargs( ) for a of couple years now and am satisfied with it in its existing state. Defining the way the command line looks and then getting switches from it is now a painless process. **DDJ**

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 192**.

C Chest **(Text begins on page 28)**

Listing One

```
/*      Getargs.h       Typedefs and defines needed for getargs
 */

#define INTEGER         0
#define BOOLEAN         1
#define CHARACTER       2
#define STRING          3
#define PROC            4

typedef struct
{
        unsigned        arg   : 7  ;    /* Command line switch          */
        unsigned        type : 4   ;    /* variable type                */
        int             *variable ;     /* pointer to variable          */
        char            *errmsg   ;     /* pointer to error message     */
}
ARG;
```

End Listing One

Listing Two

```
/*      GETARGS.C       Command line argument processor for C programs
 *
 *         (C) Copyright 1985, Allen I. Holub.  All rights reserved.
 *        This program may be copied for personal, non-profit use only.
 */

#include <stdio.h>
#include <getargs.h>

typedef int     (*PFI)();

static char     *setarg( argp, linep )
ARG             *argp;
char            *linep;
{
        /*      Set an argument. argp points at the argument table entry
         *      corresponding to *linep. Return linep, updated to point
         *      past the argument being set.
         */

        ++linep;

        switch( argp->type )
```

```
	{
	case INTEGER:
		*argp->variable = stoi( &linep );
		break;

	case BOOLEAN:
		*argp->variable = 1;
		break;

	case CHARACTER:
		*argp->variable = *linep++ ;
		break;

	case STRING:
		*(char **)argp->variable = linep ;
		linep = "";
		break;

	case PROC:
		(* (PFI)(argp->variable) )( linep );
		linep = "";
		break;

	default:
		fprintf(stderr,"INTERNAL ERROR: BAD ARGUMENT TYPE\n");
		break;
	}

	return( linep );
}

/*----------------------------------------------------------------------*/

static ARG	*findarg( c, tabp, tabsize )
int		c, tabsize;
ARG		*tabp;
{
	/*	Return pointer to argument table entry corresponding
	 *	to c (or 0 if c isn't in table).
	 */

	for(; --tabsize >= 0 ; tabp++ )
		if( tabp->arg == c )
			return tabp;

	return 0;
}
static pr_usage( tabp, tabsize )
ARG	*tabp;
int	tabsize;
{
	/*	Print the argtab in the form:
	 *		-<arg> <errmsg>     (value is <*variable>)
	 */

	for(; --tabsize >= 0 ;  tabp++ )
	{
		switch( tabp->type )
		{
		case INTEGER:
			fprintf(stderr, "-%c<num> %-40s (value is ",
						tabp->arg, tabp->errmsg);
			fprintf(stderr, "%-5d)\n", *(tabp->variable) );
			break;

		case BOOLEAN:
			fprintf(stderr,"-%c      %-40s (value is ",
						tabp->arg, tabp->errmsg);
			fprintf(stderr, "%-5s)\n", *(tabp->variable)
						 ? "TRUE": "FALSE");
			break;

		case CHARACTER:
			fprintf(stderr, "-%c<c>   %-40s (value is ",
						tabp->arg, tabp->errmsg);
			fprintf(stderr, "%-5c)\n", *(tabp->variable) );
			break;

		case STRING:
			fprintf(stderr, "-%c<str> %-40s (value is ",
						tabp->arg, tabp->errmsg);
```

(Continued on page 36)

dBASE III v

MORE POWERFUL!

dBASE III IS A *DRAMATIC* AND *POWERFUL* NEW BUSINESS PROGRAMMING LANGUAGE.
—Bob Davies, President/SBT Corporation

AFTER A TWO-HOUR HANDS-ON SESSION WITH dBASE III, IT IS *EVIDENT* THAT ASHTON-TATE HAS MADE A *SIGNIFICANT* ADVANCEMENT IN THE TECHNOLOGY OF MICROPROCESSOR-BASED DBMS.
—Robert Dew, Vice President/The Computer Society

RATHER THAN BEING AN *IMITATOR*, ASHTON-TATE HAS ONCE AGAIN SHOWN ITSELF TO BE AN *INNOVATOR*.
—Larry Heimindinger/Origin, Inc.

ASHTON-TATE HAS LISTENED TO THEIR USERS. THIS PRODUCT *ADDRESSES* EVERY ITEM ON MY *WISH-LIST*.
—Mark DaVia/National Microware, Inc.

ASHTON-TATE HAS USHERED IN A *RENAISSANCE*. THE NEXT *GENERATION* OF SOFTWARE IS NOW A *REALITY*.
—Chris MacNeil/Abel Computers

THE NEW REPORT GENERATOR IS *SUPER* TO USE AND MODIFY. LABEL GENERATION IS ALSO A NICE TOUCH.
—Michael Broska/Agate Systems, Inc.

---

dBASE III™ is the powerful and easy-to-use relational database management system you've been waiting for. You can use it without hesitation whether you're a beginner or an expert.

The big winner in the easy-to-use vs. more powerful controversy is you.

If you want to know all about dBASE III, come to your Ashton-Tate software dealer for a free demonstration. For more information call (800) 437-4329 ext. 2333 or in Colorado call (303) 799-4900 ext. 2333.

Listing Two

```
				fprintf(stderr, "<%s>)\n",
						*(char **)tabp->variable);
				break;

			case PROC:
				fprintf(stderr, "-%c<str> %-40s\n",
						tabp->arg, tabp->errmsg);
				break;
			}
		}
}
#define ERRMSG  "Illegal argument <%c>.   Legal arguments are:\n\n"

int	getargs(argc, argv, tabp, tabsize )
int	argc, tabsize ;
char	**argv ;
ARG	*tabp  ;
{
	/* Process command line arguments. Stripping all command line
	 * switches out of argv. Return a new argc. If an error is found
	 * exit(1) is called (getargs won't return) and a usage message
	 * is printed showing all arguments in the table.
	 */

	register int	nargc	    ;
	register char	**nargv, *p ;
	register ARG	*argp	    ;

	nargc = 1 ;
	for(nargv = ++argv ; --argc > 0 ; argv++ )
	{
		if( **argv != '-' )
		{
			*nargv++ = *argv ;
			nargc++;
		}
		else
		{
			p = (*argv) + 1 ;

			while( *p )
			{
				if(argp = findarg(*p, tabp, tabsize))
					p = setarg( argp, p );
				else
				{
					fprintf(stderr, ERRMSG, *p );
					pr_usage( tabp, tabsize );
					exit( 1 );
				}
			}
		}
	}
	return nargc ;
}
```

End Listing Two

Listing Three

```
/*	ARGTEST.C	Test program for getargs.
 */

#include <stdio.h>
#include "getargs.h"

int	boolarg = 0;			/* Variables used by argtab */
int	chararg = '.';
int	intarg	= 0;
char	*strarg	= "doo wha" ;
extern	proc();

ARG	Argtab[]=
```

(Continued on page 38)

Listing Three

```
{
        { 'b',  BOOLEAN,   &boolarg,         "boolean argument"          },
        { 'c',  CHARACTER, &chararg,         "character argument"        },
        { 'n',  INTEGER,   &intarg,          "integer argument"          },
        { 's',  STRING,    (int *)&strarg,   "string argument"           },
        { 'p',  PROC,      (int *)&proc,     "procedure argument"        }
};

#define TABSIZE ( sizeof(Argtab) / sizeof(ARG) )

proc( str )
char    *str;
{
        printf("Inside procedure called by -p command line switch, ");
        printf("string = <%s>\n", str );
}

main(argc, argv)
int     argc;
char    **argv;
{
        register int    i;

        printf("Argc == %d. ", argc);
        printf("Cmd line: argtest ");
        for( i = 1 ; i < argc ; printf("%s ", argv[i++]) )
                ;
        printf("\n");

        argc = getargs( argc, argv, Argtab, TABSIZE) ;

        printf("Argc == %d. ", argc);
        printf("Cmd line: argtest ");
        for( i = 1 ; i < argc ; printf("%s ", argv[i++]) )
                ;
        printf("\n");
}
```

End Listing Three

Listing Four

```
/* STOI.C       More powerful version of atoi.
 *
 *      Copyright (C) 1985 by Allen  Holub.  All rights reserved.
 *      This program may be copied for personal, non-profit use only.
 */

#define islower(c)      ( 'a' <= (c) && (c) <= 'z' )
#define toupper(c)      ( islower(c) ? (c) - ('a' - 'A') : (c) )

int             stoi(instr)
register char   **instr;
{
        /*      Convert string to integer. If string starts with 0x it is
         *      interpreted as a hex number, else if it starts with a 0 it
         *      is octal, else it is decimal. Conversion stops on encounteri
         *      the first character which is not a digit in the indicated
         *      radix. *instr is updated to point past the end of the number
         */

        register int    num  = 0 ;
        register char   *str      ;
        int             sign = 1 ;

        str = *instr;

        while(*str == ' '  ||  *str == '\t'  ||  *str == '\n' )
                str++ ;

        if( *str == '-' )
        {
                sign = -1 ;
                str++;
```

```
        }

        if(*str == '0')
        {
                ++str;
                if (*str == 'x'  ||  *str == 'X')
                {
                        str++;
                        while(  ('0'<= *str && *str <= '9') ||
                                ('a'<= *str && *str <= 'f') ||
                                ('A'<= *str && *str <= 'F')  )
                        {
                                num *= 16;
                                num += ('0'<= *str && *str <= '9') ?
                                        *str - '0'                 :
                                        toupper(*str) - 'A' + 10   ;
                                str++;
                        }
                }
                else
                {
                        while( '0' <= *str  &&  *str <= '7' )
                        {
                                num *= 8;
                                num += *str++ - '0' ;
                        }
                }
        }
        else
        {
                while( '0' <= *str  &&  *str <= '9' )
                {
                        num *= 10;
                        num += *str++ - '0' ;
                }
        }

        *instr = str;
        return( num * sign );
}
```

End Listings

Using Decision Variables in Graphics Primitives

by Tom Hogan

"Graphics algorithms" is not a listing you'll find in the* Collected Algorithms of the ACM, *but graphics programming presents unique problems to the software developer.

Picture a vast subterranean cavern packed with robed, grey-bearded men. A small platform stands at one end. Billows of purple smoke appear and out of the mist steps a tall man with white hair. He speaks and his voice fills the cavern.

"Welcome, brethren. Tonight we shall consider how to guarde our Realme of Graphix from the uninitiated who would learn our Mysteries and divulge them. We will speake of Master Bresenham, who revealed to us the Secrete of ye Mystik Circle."

Does the idea of graphics wizards jealously guarding their secrets seem strange? I can tell you that it didn't seem so to me when I searched the literature for a routine that would map ellipses and couldn't find one.

Light in the Tunnel

Thanks to a tip, I learned about *Fundamentals of Interactive Computer Graphics* by James D. Foley and Andries Van Dam (Addison Wesley, 1982). It contains an algorithm for plotting a circle using a decision variable (based on Bresenham's algorithm). I used the decision variable method (hereafter referred to as the DV method) to derive the algorithm for plotting ellipses that is found in Listing One (page 45).

The DV Method Generalized

Although the DV method was used only to plot an ellipse, it can be used to plot other conic sections as well. In fact, the DV method can be used to generate any well-behaved curve that can be expressed as $G(x, y) = 0$.

The DV Method Compared

Speed is important in graphics applications. The DV method is fast because it adds, shifts, subtracts, and multiplies integers, taking much less time than would be required by floating-point calculations.

An obvious method for plotting an ellipse is to calculate each point independently by incrementing x in unit steps and calculating y using square roots. This method is slow and generates a curve that is nonuniform when the slope of its tangent line falls below -1 (see Figure 1, page 41).

In contrast to the preceding method, the DV method does not calculate points independently. Instead, each succeeding point is mapped with reference to the point previously mapped. Furthermore, this method guarantees that succeeding points are adjacent. Therefore, the generated curve is more uniform than the curve generated by calculating points independently (see Figure 2, page 41).

Reduce the Number of Sample Points

Eight adjacent points surround any previously mapped point. In order to use the DV method to map the curve, you must reduce the number of sample points from eight to two. The sign of the decision variable can only be posi-

---

Tom Hogan, C Source, 12801 Frost Road, Kansas City, MO 64138.

tive or negative. The two possibilities correspond to the two sample points.

Take the case of an ellipse mapped only in the first quadrant. Points 1, 2, 3, 4, and 6 (see Figure 3, page 42) can be discarded immediately when mapping from Point A to C (see Figure 4, page 42), because x never decreases and y never increases as the ellipse is mapped in that direction. Points 1 through 3 would require y to increase, while points 1, 4, and 6 would require x to decrease.

This reduces the choice to three points: 5, 7, and 8. Which two are chosen as the set of sample points depends on the slopes of the line segments joining them and the last point plotted: the slope of the tangent line must fall within the range of the slopes of the two line segments for the region plotted. The slopes of the line segments of any two points must approach the slope of the tangent line as closely as possible. Thus, a set comprising Points 5 and 7 would be excluded because it would violate this requirement in the region close to B in Figure 4. There remain the possible combinations 5 and 8 and 7 and 8. Points 5 and 8, corresponding to Points S and T in Figure 5 (page 42), can be said to comprise set A. Points 7 and 8, corresponding to Points T and U in Figure 5, can be said to comprise set B.

Changing Sets of Sample Points

From Point A to B (Figure 4), set A is used because the slope of the tangent falls within the range delimited by the slopes of $\overline{PS}$ and $\overline{PT}$ (Figure 5). The decision variable is initialized to its value at Point A (see Equation 9 or 10 in the Table on page 44). The quantities to be added to it are based on the sample points in set A (Equations 6 and 8).

When the slope of the tangent line reaches −1 (at Point B in Figure 4), set B is used (see Figure 6, page 42). The value of the decision variable is recalculated (Equation 11) at Point B for set B. The quantities that are added to the decision variable are now based on set B (Equations 13 and 15).

Expansion on General Applications

Please note a key feature of the DV method: the choice of the sample points depends on the slope of the tangent line and the direction of its change. A curve can be broken up into sections for plotting, based on the range of the slope of the tangent line in each section. The method is useful for a broad spectrum of curves.

Example: The Ellipse

The initial value of y depends on the position of the major axis. If it is vertical, y is R_m, half the length of the major axis. If horizontal, y equals R_m multiplied by the aspect ratio. The positions of the major and minor axes depend on the aspect ratio. If it is less than one, then the major axis is horizontal. If greater than one, it is vertical.

Long variables were used in this routine instead of integers or doubles in order to provide the widest range of values for the aspect ratio and R_m while retaining reasonable plotting speed. Integers limited the ellipse's size too much, while doubles were too slow.

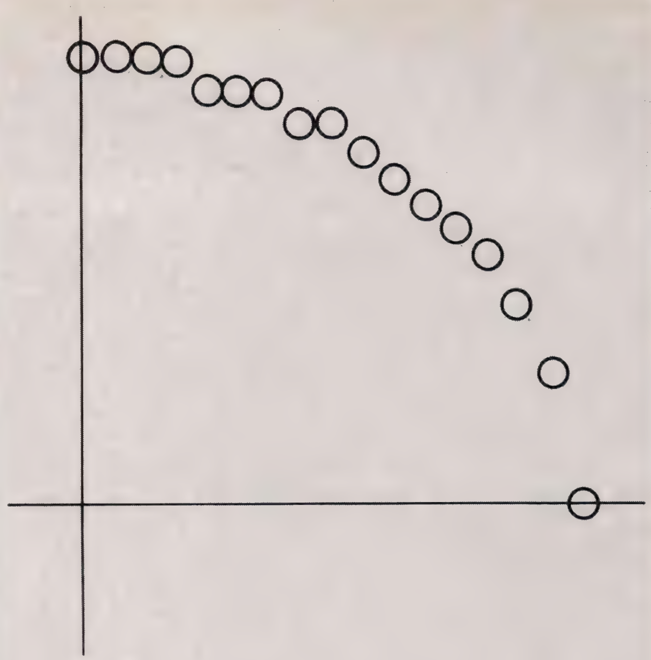


Figure 1

Circle generated by calculating points independently.

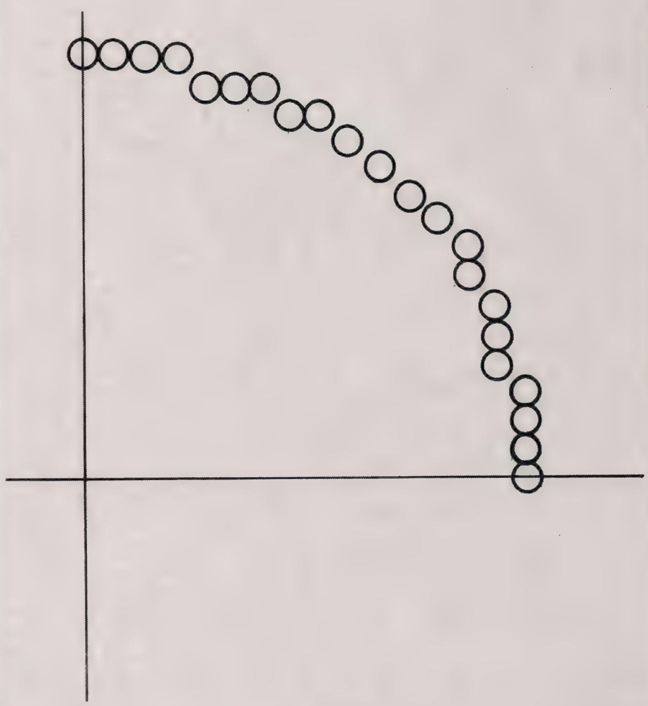


Figure 2

Circle generated by the Decision Variable Method.

The lower limit for the aspect ratio is 0.004 and R_m must not exceed 154. Otherwise, the values of some of the long variables will overflow.

This algorithm takes advantage of the symmetry of an ellipse. The ellipse is symmetric about the x and y axes, and about its center. Therefore, an ellipse need only be mapped in the first quadrant. The other three quadrants are easily plotted.

Clipping is done by the routine in Listing Two (page 48). The algorithm assumes medium resolution. **DDJ**

(Listings begin on page 45)

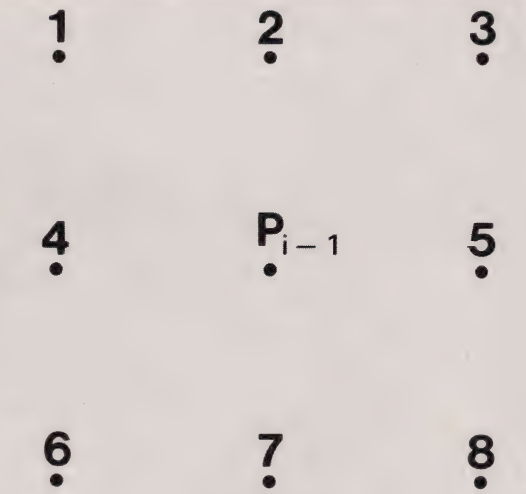


Figure 3
Last point plotted with eight adjacent points.

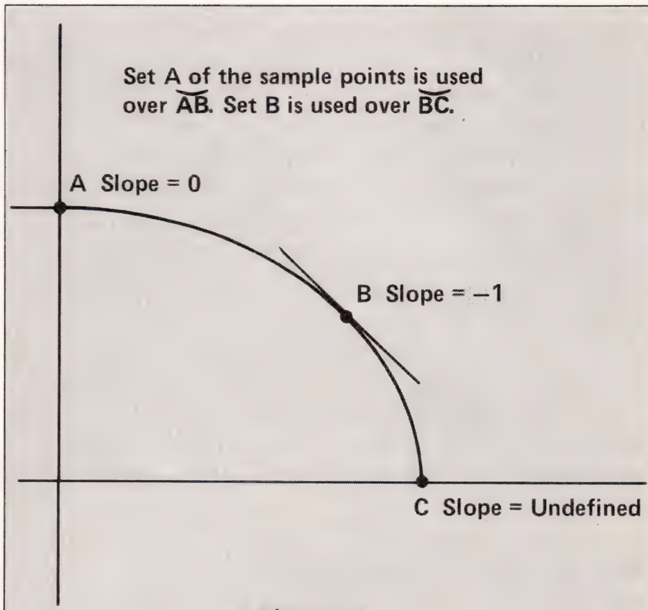


Figure 4
First quadrant of an ellipse.

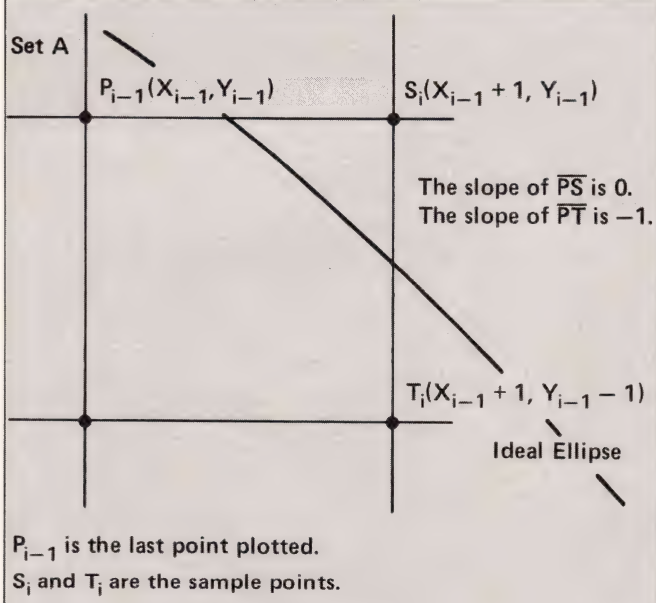


Figure 5
Last point plotted and sample points in the region where the slope of the tangent line is greater than −1.

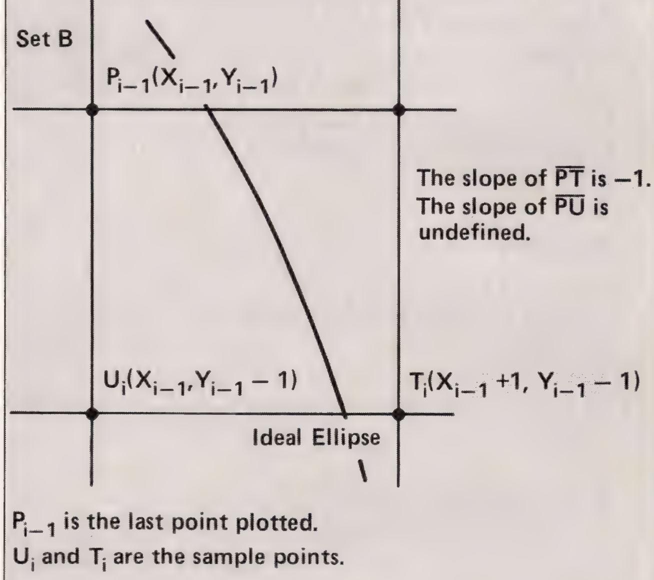
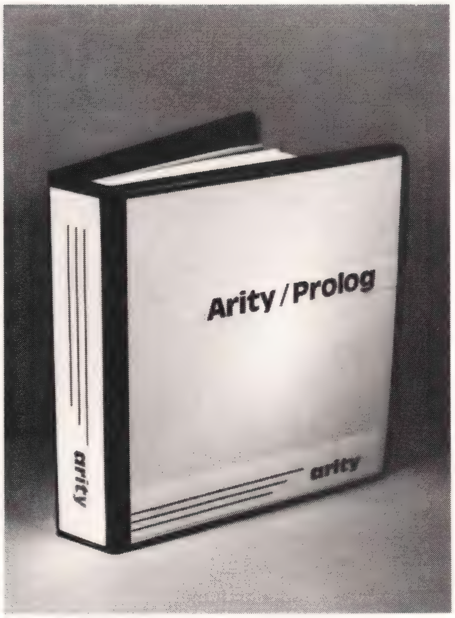


Figure 6
Last point plotted and sample points in the region where the slope of the tangent line is less than or equal to −1.

More Power Than You Thought Possible

Arity offers the first serious implementation of Prolog for IBM personal computers. Arity/Prolog is a powerful, highly optimized, and extended version of the logic programming language Prolog. Imagine building software applications with a language that solves problems through deduction and logical inference. The task of creating complex programs is much faster and easier, resulting in lower development costs. Arity/Prolog is now in use in a wide range of applications in industry, business, research, and education. The solution—the *Arity/Prolog Interpreter:*

- Source level debugger
- Virtual databases, each with a workspace of 16 megabytes
- Floating-point arithmetic
- String support for efficient text handling



- Interface to assembly language and 'C'
- Text screen manipulation
- Integrated programming shell to MSDOS
- Comprehensive set of evaluable predicates
- Definite clause grammar support

Arity/Prolog Interpreter $495.00

Arity also offers the *Arity/Prolog Compiler and Interpreter,* a sophisticated development environment for building AI applications. Essential for producing fast, serious production code.

Arity/Prolog Compiler and Interpreter $1950.00

The *Arity/Prolog Demo Disk* is available for $19.95. ■ Arity/Prolog products run on the IBM PC, XT, AT, and all IBM compatibles. ■ To order, call (617) 371-2422 or use the order form below.

arity corporation 358 Baker Avenue, Concord, MA 01742

Name ______________________

Organization ______________________

Address ______________________

Quantity	Product	Unit Price	Total Price
	Arity/Prolog Compiler & Interpreter	$1950.00	
	Arity/Prolog Interpreter	$ 495.00	
	Arity/Prolog Demo Disk	$ 19.95	
		Subtotal	
		MA residents add 5% sales tax	
		Total Amount	

☐ Enclosed is a check or money order to Arity Corporation

☐ Please bill my ☐ Mastercard ☐ Visa ☐ American Express

Account # ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Valid ___/___ to ___/___ ______________________ signature

☐ Please send me more information about Arity and Arity/Prolog

arity 358 Baker Avenue, Concord, MA 01742

M-AD-0

Circle **no. 16** on reader service card.

The well known equation of an ellipse is:

$$\alpha Y^2 + \beta X^2 - \alpha\beta = 0. \qquad \text{Eq. 1.}$$

Therefore define an error term $D(P_i)$ such that at any sample point P_i along the curve:

$$D(P_i) = \alpha Y^2 + \beta X^2 - \alpha\beta. \qquad \text{Eq. 2}$$

Define two points S_i and T_i relative to P_{i-1} such that P_i will be selected from one of these two points (see Figure 5). Define a decision variable d_i such that:

$$d_i = D(S_i) + D(T_i). \qquad \text{Eq. 3}$$

Notice that if the curve passes midway between S_i and T_i, $d_i = 0$. Further notice that:

1) if the curve passes above the midpoint, $d_i < 0$, and
2) if the curve passes below the midpoint, $d_i > 0$.

Therefore:

1) if $d_i < 0$, we choose S_i, and
2) if $d_i >= 0$, we choose T_i.

The quantity to add to d_i, depending on whether S_i or T_i is chosen, must be determined. The value of d_i for S_i and T_i must first be determined:

$$d_i = D(S_i) + D(T_i).$$
$$d_i = \alpha Y^2_{i-1} + \beta(X_{i-1} + 1)^2 - \alpha\beta + \alpha(Y_{i-1} - 1)^2 + \beta(X_{i-1} + 1)^2 - \alpha\beta.$$
$$d_i = \alpha Y^2_{i-1} + \alpha(Y_{i-1} - 1)^2 + 2\beta(X_{i-1} + 1)^2 - 2\alpha\beta. \qquad \text{Eq. 4}$$

Determine d_{i+1} if S_i is chosen:

$$d_{i+1} = \alpha Y^2_{i-1} + \beta(X_{i-1} + 2)^2 - \alpha\beta + \alpha(Y_{i-1} - 1)^2 + \beta(X_{i-1} + 2)^2 - \alpha\beta.$$
$$d_{i+1} = \alpha Y^2_{i-1} + \alpha(Y_{i-1} - 1)^2 + 2\beta(X_{i-1} + 2)^2 - 2\alpha\beta. \qquad \text{Eq. 5}$$

Define Δ_s, the difference between d_{i+1} and d_i if S_i is chosen:

$$\Delta_s = d_{i+1} - d_i.$$
$$\Delta_s = 2\beta(X_{i-1} + 2)^2 - 2\beta(X_{i-1} + 1)^2.$$
$$\Delta_s = 2\beta(2X_{i-1} + 3).$$
$$\Delta_s = 4\beta X_{i-1} + 6\beta. \qquad \text{Eq. 6}$$

Now calculate the quantity to be added to d_i if T_i is chosen:

$$d_{i+1} = \alpha(Y_{i-1} - 1)^2 + \beta(X_{i-1} + 2)^2 - \alpha\beta + \alpha(Y_{i-1} - 2)^2 + \beta(X_{i-1} + 2)^2 - \alpha\beta.$$
$$d_{i+1} = \alpha(Y_{i-1} - 1)^2 + \alpha(Y_{i-1} - 2)^2 + 2\beta(X_{i-1} + 2)^2 - 2\alpha\beta. \qquad \text{Eq. 7}$$

Define Δ_T, the difference between d_{i+1} and d_i when T_i is chosen:

$$\Delta_T = d_{i+1} - d_i.$$
$$\Delta_T = 2\beta(X_{i-1} + 2)^2 - 2\beta(X_{i-1} + 1)^2 + \alpha(Y_{i-1} - 2)^2 - \alpha Y^2_{i-1}.$$
$$\Delta_T = 4\beta X_{i-1} + 6\beta - 4\alpha Y_{i-1} + 4\alpha. \qquad \text{Eq. 8}$$

Now the value of d_i for the initial point, which is located on the Y-axis, must be determined. R_m is half the length of the major axis. If the Y-axis is the major axis, the initial point is the point $P(0, R_m)$. Therefore:

$$d_i = 2\alpha R_m^2 - 2\alpha R_m + \alpha + 2\beta - 2\alpha\beta. \qquad \text{Eq. 9}$$

If the X-axis is the major axis, the initial point is the point $P(0, (\beta/\alpha) \times R_m)$. Therefore:

$$d_i = (2\beta^2/\alpha) \times R_m^2 - 2\beta R_m + \alpha + 2\beta - 2\alpha\beta. \qquad \text{Eq. 10}$$

Consider now the slope of a line tangent to the ellipse. Beginning with the original point, it can be guaranteed that the slope of the tangent line will always decrease while the ellipse is being mapped in the first quadrant. Therefore, once the slope of the tangent line reaches -1, it will never be greater than -1. Also, it may be seen that from the point where the slope of the tangent line reaches -1 throughout the rest of the first quadrant, for any point P_{i-1} mapped, T_i will be closer than S_i. At some point, the ellipse will diverge enough from the tangent line of slope -1 for a point U_i (see Figure 6) to be closer to the ellipse than T_i. It will be sufficient to check the new set of points (T_i and U_i) from the point where the tangent line's slope becomes -1. d_i must be recalculated, as must Δ_T and Δ_U. Calculate d_i:

$$d_i = D(T_i) + D(U_i).$$
$$d_i = \alpha(Y_{i-1} - 1)^2 + \beta(X_{i-1} + 1)^2 - \alpha\beta + \alpha(Y_{i-1} - 1)^2 + \beta X^2_{i-1} - \alpha\beta.$$
$$d_i = 2\alpha(Y_{i-1} - 1)^2 + \beta X^2_{i-1} + \beta(X_{i-1} + 1)^2 - 2\alpha\beta. \qquad \text{Eq. 11}$$

Determine d_{i+1} if T_i is chosen:

$$d_{i+1} = 2\alpha(Y_{i+1} - 2)^2 + \beta(X_{i-1} + 1)^2 + \beta(X_{i-1} + 2)^2 - 2\alpha\beta. \qquad \text{Eq. 12}$$

Define Δ_T as the difference between d_i and d_{i+1} if T_i is chosen:

Table
Derivation of the algorithm to map an ellipse.

$$\begin{aligned}\Delta_T &= d_{i+1} - d_i.\\ &= 2\alpha(-2Y_{i-1} + 3) + \beta(4X_{i-1} + 4).\end{aligned} \qquad \text{Eq. 13}$$

Determine d_{i+1} if U_i is chosen:

$$\begin{aligned}d_{i+1} &= 2\alpha(Y_{i-1} - 2)^2 + \beta X^2_{i-1}\\ &+ \beta(X_{i-1} + 1)^2 - 2\alpha\beta.\end{aligned} \qquad \text{Eq. 14}$$

Define Δ_U as the difference between d_{i+1} and d_i if U_i is chosen:

$$\begin{aligned}\Delta_U &= d_{i+1} - d_i.\\ &= 2\alpha(-2Y_{i-1} + 3).\end{aligned} \qquad \text{Eq.15}$$

The point where the slope becomes less than -1 must be determined. Finding the first derivative of the ellipse produces the equation:

$$\alpha YY' = -\beta X. \qquad \text{Eq. 16}$$

When the slope is required to be less than -1, the relation

$$\alpha Y < \beta X \qquad \text{Eq. 17}$$

determines the point at which this occurs. This concludes the derivation of the algorithm for mapping an ellipse.

Decision Variables (Text begins on page 40)

Listing One

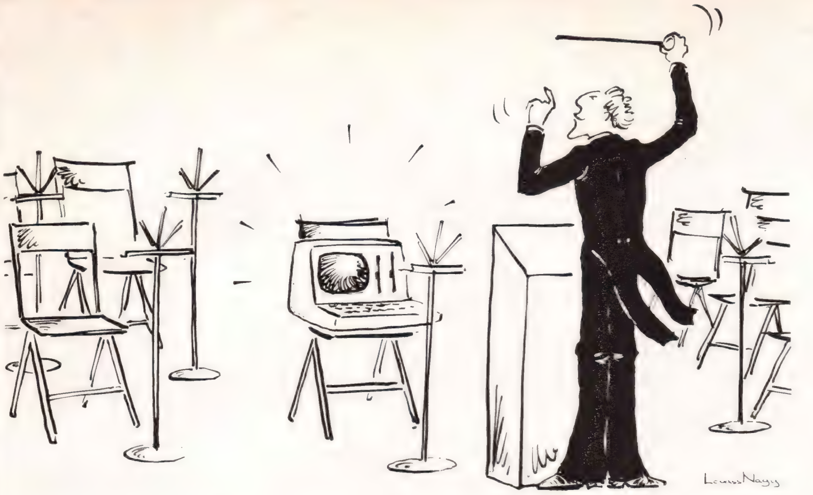
```
/*   compiled 11/28/84 under Lattice C Compiler ver. 2.13
 *
 *   ellipse.c
 *     by Tom Hogan
 *   Version 2.0
 *   Copyright 1984
 *   by C Source Inc.
 *   All rights reserved.
 *   Permission is granted
 *   for unlimited
 *   personal, non-
 *   commercial use only.
 *
 *   For answers to questions,
 *   please write to:
 *
 *   Tom Hogan
 *   C Source Inc.
 *   12801 Frost Road
 *   Kansas City, MO 64138
 *
 *   If you would rather call,
 *   our number is (816) 353-8808.
 *
 *   Please call between 9 AM
 *   and 5 PM CST, Monday thru
 *   Friday.  Calls can only
 *   be returned on a collect basis.
 */

#define ERROR  0
#define SUCCESS 1

#define MAX_COL  319
#define MAX_ROW  199
#define MIN_COL  0
#define MIN_ROW  0

            /* a macro used in the error-checking code */

#define inrange(a,x,b)    ((a) <= (x) && (x) <= (b))

      /* the bounds of the following array can be set for different   *
       * graphics modes; this array is used in the assembly algorithm *
       * Listing 2 to write individual pixels.                        */
```

(Continued on next page)

Listing One

```
int CRT_BNDS[] = { MAX_COL, MAX_ROW, MIN_COL, MIN_ROW };

          /* r_sub_m is half the length of the major axis */

ellipse(x, y, r_sub_m, color, aspect)
     int x, y, r_sub_m, color;
     double aspect;
{
int two_x, two_y, or_color;
int col, row, rel_x, rel_y;  /* rel_x and rel_y are relative */
double square_aspect;        /* to the center               */
long temp, beta, alpha, two_alpha, four_alpha, two_beta, four_beta, d;

                    /* d is the decision variable */

                    /*   error checking follows   */

if ((aspect < 0.004) || !inrange(0, color, 15) || !inrange(1, r_sub_m,
     154)) {
     printf("ELLIPSE(%d, %d, %d, %d, %f) - BAD ARG\n", x, y, r_sub_m,
         color, aspect);
     return ERROR; }
square_aspect = aspect * aspect;      /* initialize the beginning row  */
if (aspect < 1.0) {                   /* and set the values of         */
     alpha = r_sub_m * r_sub_m;       /* constants                     */
     beta = alpha * square_aspect;
     row = y + r_sub_m * aspect; }
else {
     beta = r_sub_m * r_sub_m;
     alpha = beta / square_aspect;
     row = y + r_sub_m; }
if (alpha == 0L) alpha = 1L;     /* compensates for very small ellipses */
if (beta == 0L) beta = 1L;
or_color = 0xc00 | color; /* sets the high byte for a screen interrupt */
col = x;                  /*     initializes the beginning column      */
two_x = x << 1;
two_y = y << 1;
rel_y = row - y;
two_alpha = alpha << 1;
four_alpha = alpha << 2;
four_beta = beta << 2;
two_beta = beta << 1;

            /* initialize the decision variable -- Eq. 9 or 10 */

d = two_alpha * ((rel_y - 1) * rel_y) + alpha + two_beta * (1 - alpha);

          /* when the slope <= -1, choose the new set of points */
          /*    -- Eq. 17                                       */

 while (alpha * (rel_y = row - y) > beta * (rel_x = col - x)) {
      write_pix(col, row, two_x - col, two_y - row, or_color);
      if (d >= 0) {
           d += four_alpha * (1 - rel_y);     /* Eq. 6; also first    */
           row--; }                           /* half of Eq. 8;       */
      d += two_beta * (3 + (rel_x << 1));     /* second half of Eq. 8 */
      col++; }

 /* initialize the decision variable for the rest of the ellipse */
 /*    -- Eq. 11                                                 */

 d = two_beta * (rel_x * (rel_x + 1) + two_alpha * (rel_y * (rel_y - 2)
        + 1) + (1 - two_alpha) * beta;

 while ((rel_y = row - y) + 1) {
      write_pix(col, row, two_x - col, two_y - row, or_color);
      if (d <= 0) {                           /* col - x == rel_x     */
           d += four_beta * (1 + col - x);    /* Eq. 15; also first   */
           col++; }                           /* half of Eq. 13;      */
      row--;
```

(Continued on page 48)



Would you hire an entire band when all you need is one instrument? Of course not.

So why use a whole orchestra of computers when all you need is one to develop software for virtually any type of micro-processor?

The secret? Avocet's family of cross-assemblers. With Avocet cross-assemblers you can develop software for practically every kind of processor — *without having to switch to another development system along the way!*

Cross-Assemblers to Beat the Band!

Development Tools That Work

Avocet cross-assemblers are fast, reliable and user-proven in over 4 years of actual use. Ask NASA, IBM, Xerox or the hundreds of other organizations that use them. Every time you see a new microprocessor-based product, there's a good chance it was developed with Avocet cross-assemblers.

Avocet cross-assemblers are easy to use. They run on almost any personal computer and process assembly language for the most popular microprocessor families.

Avocet Cross-assembler	Target Microprocessor	CP/M-80	CP/M-86 IBM PC, MSDOS**
XASM04 *NEW*	6804	$ 250.00	$ 250.00
XASM05	6805	200.00	250.00
XASM09	6809	200.00	250.00
XASM18	1802/1805	200.00	250.00
XASM48	8048/8041	200.00	250.00
XASM51	8051	200.00	250.00
XASM65	6502/65C02	200.00	250.00
XASM68	6800/01, 6301	200.00	250.00
XASM75	NEC 7500	500.00	500.00
XASM85	8085	250.00	250.00
XASM400	COP400	300.00	300.00
XASMF8	F8/3870	300.00	300.00
XASMZ8	Z8	200.00	250.00
XASMZ80	Z80	250.00	250.00
XMAC682 *NEW*	68200	595.00	595.00
XMAC68K *NEW*	68000/68010	595.00	595.00

Your Computer Can Be A Complete Development System

Avocet has the tools you need to enter and assemble your soft-ware and finally cast it in EPROM:

VEDIT Text Editor makes source code entry a snap. Full-screen editing plus a TECO-like command mode for advanced tasks. Easy installation - INSTALL program supports over 40 terminals and personal computers. Customizable keyboard layout. CP/M-80, CP/M-86, MSDOS, PCDOS $150

EPROM Programmers let you program, verify, compare, read, display EPROMS but cost less because they communicate through your personal computer or terminal. No personality modules! On-board intelligence provides menu-based setup for 34 different EPROMS, EEPROMS and MPUs (40-pin devices require socket adaptors). Self-contained unit with internal power supply, RS-232 interface, Textool ZIF socket. Driver software (sold separately) gives you access to all programmer features through your computer, lets you download cross-assembler output files, copy EPROM to disk.

Model 7228 Advanced Programmer — Supports all PROM types listed. Super-fast "adaptive" programming algorithm programs 2764 in 1.1 minutes.

Model 7128 Standard Programmer — Lower-cost version of 7228. Supports all PROM types except "A" versions of 2764 and 27128. Standard programming algorithm programs 2764 in 6.8 minutes.

Model 7956 and 7956-SA Gang Programmers — Similar features to 7228, but program as many as 8 EPROMS at once. 7956-SA stand-alone version copies from a master EPROM. 7956 lab version has all features of stand-alone plus RS-232 interface.

EPROM: 2758, 2716, 2732, 2732A, 2764, 2764A, 27128, 27128A, 27256, 2508, 2516, 2532, 2564, 68764, 68766, 5133, 5143. **CMOS:** 27C16, 27C32, 27C64, MC6716. **EEPROM:** 5213, X2816A, 48016, I2816A, 5213H. **MPU (w/adaptor):** 8748, 8748H, 8749, 8749H, 8741, 8742, 8751, 8755.

7228	**Advanced Programmer**	**$ 549**
7128	**Standard Programmer**	**429**
7956	**Laboratory Gang Programmer**	**1099**
7956-SA	**Stand-Alone Gang Programmer**	**879**
GDX	**Driver Software**	**95**
481	**8748 Family Socket Adaptor**	**98**
511	**8751 Socket Adaptor**	**174**
755	**8755 Socket Adaptor**	**135**
CABLE	**RS-232 Cable (specify gender)**	**30**

HEXTRAN Universal HEX File Converter — Convert assembler output to other formats for downloading to development systems and target boards. Also useful for examining object file, changing load addresses, extracting parts of files. Converts to and from Intel, Motorola, MOS, RCA, Fairchild, Tektronix, TI, Binary and HEX/ASCII Dump formats. For CP/M, CP/M-86, MSDOS, PCDOS $250

Ask about UNIX.

68000 CROSS-ASSEMBLER — With exhaustive field testing completed, our 68000 assembler is available for immediate shipment. XMAC68K supports Motorola standard assembly language for the 68000 and 68010. Macros, cross-reference, structured assembly statements, instruction optimization and more. Linker and librarian included. Comprehensive, well-written manual.

To find out more, call us toll-free.

1-800-448-8500

(in the U.S. Except Alaska and Hawaii)

VISA and Mastercard accepted. All popular disc formats now available --please specify. Prices do not include shipping and handling -- call for exact quotes. OEM INQUIRIES INVITED.

*Trademark of Digital Research **Trademark of Microsoft

AVOCET SYSTEMS INC.™

Sales and Development: 585-DDJ
10 Summer Street
P.O. Box 490, Dept.
Rockport, Maine 04856
(207) 236-9055 Telex: 467210 AVOCET CI

Corporate Offices:
804 South State Street
Dover, Delaware 19901

Decision Variables (Listing Continued, text begins on page 40)

Listing One

```
    d += two_alpha * (3 - (rel_y << 1)); }  /* second half of Eq. 13 */
return SUCCESS; }

         /*  write_pix writes four points to the screen, taking *
          *  advantage of the ellipse's four-way symmetry       */

write_pix(col, row, neg_col, neg_row, or_color)
     int col, row, neg_col, neg_row, or_color;
{
_pixasm(col, neg_row, or_color);             /* symmetry -- y axis */
_pixasm(neg_col, neg_row, or_color);         /* symmetry -- center */
_pixasm(neg_col, row, or_color);             /* symmetry -- x axis */
_pixasm(col, row, or_color); }               /* mapped point       */
```

End Listing One

Listing Two

8086 Assembly function that clips against the screen bounds

```
                        ;        _pixasm(x, y, type_color)
                        ;             int x, y, type_color
                        ;        type  - get or read pix in hi byte
                        ;        color - if set pix, in lo byte
                        ;
 DGROUP GROUP     DATA
 DATA   SEGMENT   WORD PUBLIC 'DATA'
        ASSUME    DS:DGROUP

        EXTRN CRT_BNDS:WORD

 PGROUP GROUP      PROG
 PROG   SEGMENT    BYTE PUBLIC 'PROG'
        ASSUME     CS:PGROUP

        PUBLIC _pixasm

 _pixasm proc near
           push      bp               ; set return address
           mov       bp, sp

           mov       cx, [bp+4]       ; set x (column)
           cmp       cx, CRT_BNDS     ; check for max x value
           jg        BYE
           cmp       cx, CRT_BNDS+4   ; check for min x value
           jl        BYE
           mov       dx, [bp+6]       ; set y (row)
           cmp       dx, CRT_BNDS+2   ; check for max y value
           jg        BYE
           cmp       dx, CRT_BNDS+6   ; check for min y value
           jl        BYE
           mov       ax, [bp+8]       ; set color (al) / fcn number (ah)
           int       10h              ; do video interrupt
           sub       ah, ah
BYE:       pop       bp               ; reset frame pointer
           ret                        ; bye!

 _pixasm endp
 PROG ends
 DATA ends
 end
```

End Listings

"It's Time Someone Got Serious About The CRITICAL Issues Concerning LAN Software"

"The Industry has a responsibility to the buying public to stop using MIRRORS to promote VAPORWARE claims about multi-user software capabilities. Many complex and potentially DANGEROUS situations exist for any software that runs in a multi-user / LAN environment. Partial solutions may be acceptable when using word processing or spreadsheet applications, however Data Base Management Systems demand complete solutions. Unlimited Processing Incorporated has devoted millions of dollars and three and one half years developing the expertise necessary to present a true interactive multi-user DBMS and APPLICATION DEVELOPMENT ENVIRONMENT.

George Treiber
President, UP, Inc.

To evaluate "multi-user claims," you must ask the following questions:

- **IS IT REALLY MULTI-USER, NOT MERELY RESTRICTED SINGLE-USER?**
  INFORMA Is!
  Can an unlimited number of simultaneous users:
  Access the same file?
  Access the same file in different sort orders?
  Access the same record?
  Update the same record?
  Enter new records into the same file?
  Does the system prevent one user from changing the appearance of the database while another user is reviewing that data?

- **DOES IT REALLY MANAGE YOUR DATABASE, NOT JUST STORE DATA?**
  INFORMA Does!
  With complete transparency to users does it:
  Allow data files to reside on different disk drives?
  Keep track of the existence and location of data files for all applications?
  Eliminate data compression and other time consuming operations?
  Permit data file movement for disk balance?

- **IS DATA INTEGRITY GUARANTEED, INSTEAD OF ACCEPTING DATA LOSS AS AN INEVITABLE RISK?**
  INFORMA Guarantees!
  Can you be assured one update will not overwrite another when two or more users simultaneously:
  Change the same record?
  Change adjacent records?

- **IS COMPLETE SECURITY EMBEDDED THROUGHOUT THE SYSTEM, NOT MERELY AN AFTERTHOUGHT?**
  It Is With INFORMA
  Are multiple levels of user security offered?
  Can each activity be secured by specific user?
  Can applications be secured by specific user?
  Can audit trails be created by user and application?
  Can the functions find, enter, update, delete, and print be secured by user and application?

- **IS IT FAST ENOUGH TO BE EFFECTIVE, NOT TOO SLOW TO BE USEFUL?**
  INFORMA Has Unequaled Speed!
  Retrieve any record in less than 1 sec?
  Does the access time remain relatively constant:
  As the number of records increases?
  As the number of users increases?

INFORMA offers the only serious solution to Interactive, Real-Time, DataSharing.

UNLIMITED PROCESSING INCORPORATED

8382 Baymeadows Road, Suite 8
Jacksonville, Florida 32216
(904) 731-8330 and (800) 874-8555
Telex 350754 (800) 874-4185

Introductory Offer

Single-user	LAN/Multi-user
$199	**$599**
regularly $795	regularly $1495

Offer Expires April 10, 1985

Circle **no. 87** on reader service card.

Solid Shape Drawing on the Commodore 64

by Richard Rylander

Computer graphics is one of the fastest growing areas of computer software and hardware development. The old saying that "one picture is worth a thousand words" couldn't be truer when you consider the effectiveness of a pie chart or bar graph versus columns of numbers. Even the simplest of home computers is capable of at least this basic form of computer graphics. But the term *computer graphics* usually brings to mind more exciting images than just mundane business applications.

Special visual effects for motion pictures and some television advertisements are now being done with computers to create scenes of "enhanced reality." We see spacecraft making impossible manuevers or shimmering corporate logos "flying by" in ways that would be difficult to simulate with conventional photographic techniques. The enormous amount of data and processing necessary to produce such images has made the use of "super computers" essential. Even with the largest computers working full time on the task, only a few feet of film are produced per day. For home computer owners who want to create their own computer-generated masterpiece, however, the task is not hopeless.

The Apple Macintosh represents a major step in bringing powerful, easy-to-use graphics capabilities to the masses. The "masses," however, may still find the Macintosh pricey, especially if they just want to play around a bit with MacPaint. Low-cost computers, such as the Commodore 64, while lacking the slick graphic interaction of the Mac, are selling for such ridiculously low prices ($150 at the time of writing this article) that they are hard to pass up. They have the potential for some fairly sophisticated graphics, but so far most of the commercially available software has done little more than add the usual "point plotting" and "line drawing" commands.

This article describes a software package that will allow you to create your own computer-generated scenes, such as "Coffee and Donuts" in Figure 1 (page 55) and "Goblets" in Figure 2 (page 55). You can construct images from combinations of elementary shapes (spheres, cylinders, and toroids in various orientations) or by the "polygon mesh" technique typically used to render more complex objects. You can produce drawings with realistic shading effects and in a variety of styles to make some surprisingly detailed pictures with a minimum of effort. The package includes demonstration programs to illustrate how you use each graphic function and style option.

In keeping with the "running light without overbyte" theme of *DDJ*, the entire graphics package fits in 3K of RAM. The program sits in an area of

Pumping pixels requires tight, efficient code. Here are halftone shading, backlighting and other visual effects in a tight little package.

Richard Rylander, 179 N. McKnight Rd. Apt. 203, St. Paul, MN 55119.

RAM that is inaccessible to BASIC (a 4K block following the BASIC ROM) so that you don't lose any useful program space. Commodore supplies a "DOS Wedge" program that occupies the last 1K of this block, and one of my objectives in the design of the graphics package was to maintain compatibility with this useful utility.

The compactness of the code required me to leave out some "bells and whistles," but I've made no sacrifices to execution speed or user-interface ease. The main omission is error trapping—the software performs no checking to ensure that you use only "legal" point coordinates and so on. This may make program development a little more difficult, but once you have written and debugged a program properly, error checking becomes extraneous and only slows program execution.

We must address several general tasks in developing a shape-drawing program. The first is, of course, how to calculate the apparent brightness for all points on visible surfaces of objects. Next, how do we display these different brightness levels on a high-resolution display that is basically on or off? Finally, a "general" task, but detailed in nature, is creating the specific software tools that will let us do the required calculations and bit-map manipulations.

The program itself is written in machine language, because even with our simplifying restrictions, the plotting of each pixel still involves considerable computation. The program uses integer arithmetic throughout, and we utilize all the symmetry available to eliminate redundant calculations. The main program actually consists of five separate subprograms to break the process into manageable pieces and to provide a library of separate utilities that you may find useful in other programs. Before getting into the problem of shade calculation, I'll present the first subprogram, which is a collection of integer arithmetic utilities that all the later subprograms need.

Integer Arithmetic Utilities

Listing One (page 61) provides the source code for a set of four integer arithmetic routines. Because we will draw our shaded shapes on a high-resolution display of 320 x 200 points, we need only single-precision arguments for most functions and double-precision results for some intermediate values.

The first two routines, multiplying two single-precision numbers to yield a double-precision product and dividing a double-precision dividend by a double-precision divisor to yield a double-precision quotient, are fairly standard routines that should need no description other than the comments in the code. The multiplication routine also contains a special case to treat a single-precision number as a signed integer, which is then squared.

The integer-square-root routine deserves some special comment. Have you ever tried the BASIC square-root function? If you have, you know it takes about 52 milliseconds. At this rate, plotting the 64,000 pixels of a 320 x 200 screen would take about 3,328 seconds, or nearly an hour, if we need a square root per point! Of course, we don't need a square root for each point, but it is an

essential part of many shade calculations. Obviously, anything we can do to speed up this particular function will have a great effect on the overall program speed.

The BASIC square-root routine is slow because BASIC is written to save space, not time. Virtually all small BASICs find a square root by first taking the log of the argument, dividing that result by 2, and then exponentiating. Because LOG and EXP functions are already available, why use any more space than necessary to derive SQR?

One way to speed up this function is to use Newton's method, an iterative procedure that can give full floating-point precision in less time. Because we are interested only in integer results, we can do much better yet, though.

The method we actually use is essentially as fast as doing a single-divide operation. We construct the root, bit by bit, in a manner similar to the way we would take a decimal square root by hand using pencil and paper (does anyone remember how to do that in this age of electronic calculators?). The process is even easier for a binary root because the guesses are, of course, only 1 or 0. The procedure is almost identical to that for division: we guess a partial root bit (as we would guess the next quotient bit) and, in effect, compare the square of the partial quotient to the argument to see if we keep the 1 guessed or change it to a 0 and restore the argument. For more details about this procedure, I recommend section 17.3, "Binary Square Roots—Restoring Method," of the book *The Logic of Computer Arithmetic* by Ivan Flores (Englewood Cliffs, NJ: Prentice-Hall).

The last integer arithmetic routine is a fast procedure for generating pseudo-random bytes. An old and popular method for generating pseudo-random numbers is the so called congruential method, in which you derive each successive random number from the previous random number by multiplying by a suitably chosen constant and taking the product modulo P, where P is a large prime (see, for example, "A Better Random Number Generator," by H. Cem Kaner and John R. Vokey in the June 1984 issue of *MICRO*). Again, though, because we are interested only in generating random bytes, we can use a much faster routine.

The method we use is to store two previous random bytes, exclusive OR them bit by bit, and rotate the result to the right by one bit to generate a new random byte. Because it needs no multiply or divide operations, provides a fairly uniform distribution of random bytes, and has a long period, this process is fast. Note that even though we are only generating random bytes (0-255), the pattern or sequence of successive bytes doesn't repeat until after 35,805 calls to the random-number-generator routine (for the initial parameters used in the program). The long period is necessary to ensure that we won't produce any unwanted secondary patterns in what should be "randomly" shaded images—the human eye is adept at picking out such correlations.

Now that some low-level number crunching is out of the way, we can concentrate on the graphics aspects of the main program.

Graphics Utilities

Listing Two (page 62) provides the source code for the elementary graphics functions of displaying and clearing the bit-map, filling the color map (determining dot/background colors), and plotting (and unplotting) individual points both directly and with the plot or unplot decision weighted by a shade value and a particular shade style function. The PLOT (and UNPLOT), CLEAR, COLOR, GRFON (display graphics screen), and GROFF (return to text screen) routines are specific to the Commodore 64. To adapt this program for other 6502-based computers, you need to rewrite these routines, but the rest of the graphics package (save the final user interface to BASIC) is machine independent.

The SHADE routine is simple and straightforward. We set up an 8 x 8 threshold matrix as a linear array of values 0 to 63. The six bits we need to address an element of this threshold array are determined by masking off the lower three bits of the absolute X and Y screen coordinates (we take X modulo 8 and Y modulo 8) with the Y bits shifted to become the upper three bits. This effectively repeats the threshold pattern over the entire bit-map area. For each point within an object to be drawn, we calculate a shade value normalized to the range 0-63 and then compare it to the threshold value at that screen point to decide whether to plot or unplot.

If we have, for example, a large area where the shade value is constant at 31 (a 50 percent gray), then within each 8 × 8-pixel character cell block, half of the threshold values will be greater than or equal to the shade value, turning on half the pixels. The order in which the values 0 to 63 are arranged in the threshold table determines how these on pixels are distributed. A common pattern for these so called ordered-dither matrices is a recursive arrangement such as that shown in Table 1 (page 53). If that matrix is subdivided into quarters, sixteenths, and so on, each submatrix has the same general pattern, with offsets between submatrices.

The matrix in Table 1 (classic Bayer ordered dither) keeps the on and off pixels spread as far apart as possible for any shade. This keeps the spatial frequencies in the shade "texture" as high as possible for a particular shade. A disadvantage, though, is that texture changes then accompany shade changes, making shade quantization more apparent. Another disadvantage is that many color monitors have a hard time coping with the very high frequency on-off-on sequence of pixels, distorting shade values when isolated pixels get lost.

As an aside, a convenient algorithm to generate the ordered-dither matrix in Table 1 (or such a matrix of any size) is found in Figure 7 (page 59).

The threshold matrix we actually use in the program is shown in Table 2 (page 53). Here we follow a recursive scheme as we start turning on pixels until eight nuclei have been established. As shades increase, new pixels are added around the edges of these nuclei, simulating the dot-growth behavior seen in normal printing halftones. Except in the ex-

treme highlight and shadow regions, the shade texture remains fairly constant. Also, the clustering of on or off pixels is much less demanding on the display bandwidth. But take your pick—try filling the threshold matrix with your own patterns.

The RSHADE routine shades by comparing the shade value to a pseudo-random byte shifted right twice to match the 0-63 range. This scheme also tends to average out the tone errors generated as each pixel is turned only on or off (though we want an intermediate shade of gray) by dithering the threshold value randomly at each pixel. Both shading schemes produce sharp edges because each pixel is plotted independently. Abrupt shade changes are then followed faithfully.

The SCALE routine provides an opportunity to mention a general rule you should follow when you try to write fast programs: If at all possible, avoid division; and if you must divide, try to make it by a power of 2 (so that you can use right shifts). The SCALE routine helps correct some of the geometric distortion that otherwise results from plotting objects on the Commodore 64's rectangular bit map. While the monitor display has an aspect ratio of approximately 4:3, the bit-map aspect ratio is 320:200 or 8:5. A simple way to keep sphere outlines circular (instead of the usual egglike appearance) is to work in pseudoscreen coordinates of 320 × 240, giving the bitmap the same 4:3 aspect ratio as the screen display. The SCALE routine then converts 0-239 YPLOT values to a 0-199 range.

An obvious way to do this is to first multiply by 5 and then divide by 6. Or you might save a multiplication by first dividing a copy of YPLOT by 6 and subtracting that from the original YPLOT. A *much* faster way is to multiply the single-precision YPLOT by 213 and then take just the upper byte of the double-precision product (effectively dividing by 256) as the scaled YPLOT value. This gives us the proper range of absolute screen Y values and "rounds" the scaling as well.

This method of scaling means that you effectively replot every sixth Y line of pixels, but the routine's simplicity more than makes up for plotting 20 percent more points. Scaling is left as an option because some printers (such as the Commodore 1525) have 1:1 dot densities. By not scaling, the screen display is distorted but a hard-copy printout will have the proper geometry. On the other hand, you can set up printers, such as the Epson RX-80, with the appropriate horizontal and vertical dot densities to produce a 4:3 aspect ratio screen dump. Scaling here corrects both the screen and the hard-copy output.

The routine PLTSHD is a higher level routine that simply checks a couple of flags to see what kind of shading we want and whether to scale or not. It then calls the appropriate shading routine. To plot a shade-weighted pixel from BASIC, POKE a shade value (0-63) into VALUE; POKE the absolute X and (optionally scaled) Y values into XPLOT, XPLOT+1, and YPLOT; set up the flags HTORRN (HALFTONE or RANDOM) and NOSCAL; then SYS to PLTSHD. This seems like a lot of work to plot a single point, but we really won't be plotting shade-weighted points by hand—the later shape-drawing routines take care of this.

Line and Facet Drawing

Listing Three (page 64) completes the elementary graphics functions by adding line-drawing and shaded-facet-drawing routines. We draw lines using a modified form of Bresenham's algorithm, a DDA (Digital Differential Analyzer) technique that keeps the actual plotted points within one half-screen unit of the true line. Regardless of the order in which we specify the line's endpoints, the program sorts them so that lines are always drawn from left to right (the X position is incremented or unchanged at each step). We then need only determine which is greater, the change in X or the change in Y, and whether the Y coordinate difference is positive or negative. The program checks

0	32	8	40	2	34	10	42
48	16	56	24	50	18	58	26
12	44	4	36	14	46	6	38
60	28	52	20	62	30	54	22
3	35	11	43	1	33	9	41
51	19	59	27	49	17	57	25
15	47	7	39	13	45	5	37
63	31	55	23	61	29	53	21

Table 1

Recursive Arrangement of Ordered-Dither Matrix

0	8	53	61	2	10	55	63
16	24	37	45	18	26	39	47
49	57	4	12	51	59	6	14
33	41	20	28	35	43	22	30
3	11	54	62	1	9	52	60
19	27	38	46	17	25	36	44
50	58	7	15	48	56	5	13
34	42	23	31	32	40	21	29

Table 2

Threshold Matrix

the scale flag to see if the endpoints should be adjusted (in their Y coordinates) before plotting commences. This keeps a common coordinate system for drawing shaded shapes (spheres and so on) and lines in the same image. A flag MODE determines whether the line is drawn by setting or clearing pixels ("black" or "white" lines).

The program's last elementary function is to draw shaded triangular facets. A powerful and flexible technique for rendering objects is by means of a polygon mesh. Natural polyhedral objects (for example, cubes and pyramids) are obvious candidates for this method, but you can also approximate curved surfaces by a connected mesh of planar polygonal sections. As in piecewise-linear approximations of curves, the finer the segmentation, the better the approximation—although at the price of greater computational overhead. The coordinates of the polygon vertices then constitute a data base that you can manipulate easily for rotational transformations, perspective transformations, and so on. Triangular sections are the simplest to handle and the most general because you can break any higher-order polygon down into triangular sections.

As in the line-drawing routine, the program sorts the endpoints of the facets into a left-to-right order and checks the scale flag to maintain a consistent coordinate system. At each X position across the facet, a top and bottom Y coordinate pair is determined by a simple proportionality between the X difference and Y difference for the particular triangular sides. Although this involves both multiplication and division operations, it is not the innermost loop here (drawing the shaded line segment between Y pairs is). It also does not give more than one Y value per X position, as would a DDA method (for lines with slope greater than 1 in magnitude).

The maximum X difference is restricted to less than 256 (single precision), though the absolute X coordinates for the triangle vertices can be anywhere on the screen. This is not too severe a constraint because most polygon meshes are made up of small sections, and, if a larger triangle is essential, you can break it into smaller triangles that meet this condition. The occasional inconvenience this might cause is worth the increase in speed and shortened program.

We leave the shade value used in drawing facets as a parameter that the calling program specifies. Although adding "surface normal" calculation and shade value computation based on some illumination model (and requiring the Z coordinates then to be specified for the vertices) is not difficult, the same shade value is used for all points of the facet, making shade determination by a BASIC program practical. Also, leaving shade calculations up to BASIC adds the flexibility that we can use any shading model. The curved surface-drawing routines I describe later do include shade determination, but here each point can potentially have a different shade value, putting it in the innermost loop. For this reason, the curved surfaces must have a fixed shading model.

An interesting use for the BASIC-specified shade values is drawing "white" facets by setting the shade to 64. This doesn't seem particularly useful, but when you draw "wireframe" polyhedra, it makes a simple, hidden-line removal scheme possible. If the facets of an object are sorted so that they are drawn from those furthest to those nearest the observer, with "black" lines added around the edges of each facet, then foreground facets that partially obscure background facets erase the lines in their interior. We can set a flag EDGES so that, after the program has drawn a shaded facet, it can add lines automatically to outline and emphasize the edges. The MODE flag again determines how the lines are drawn (set or cleared).

Now that we have a toolbox of arithmetic and primitive-graphics routines, we can concentrate on the main subprogram for drawing shaded, curved surfaces rapidly.

Determining Shade Values for Curved Surfaces

Many recent advances in computer image synthesis have dealt specifically with how different surfaces reflect, refract, scatter, or absorb light. Increasingly complex models for the visual appearance of various surface textures have led to ever more realistic images, but increasing computational overhead accompanies these models. We'll employ the simplest shading scheme to start with and then suggest how you might modify it to add "realism," while still keeping the computations manageable.

If we restrict ourselves to simple, symmetrical objects, we can greatly simplify the problem of calculating surface brightness. This is not a severe restriction in itself, as you can break most "complex" objects down into combinations of simple elemental shapes. We will have to limit ourselves to "normal" (that is, head-on) views of the objects, because once we allow objects to be rotated, they lose most of their symmetry. Also, calculation time (particularly that for hidden surface removal) increases enormously.

We use the Lambertian (diffuse reflector) model to determine the surface brightness. We assume that light comes from a single, specific direction and then find the cosine of the angle between this reference vector and a surface normal vector. This gives us shade values from 1, where the surface faces the light source, to 0 at the terminator, where light just grazes the surface; and negative values where the surface is turned away from the light. Here is our first option: We can clip the brightness values at 0 so that areas facing away from the light have zero brightness, as an object lit by a single source (in deep space) would appear. Or, we can use the absolute value of the cosine so that the object appears to be lit by two identical sources on opposite sides. Either way is simple, so we allow for both cases by using a flag to determine what kind of illumination scheme we want to use for a whole scene or any individual shape making up the scene.

Although this simple model is valid for diffusely reflecting surfaces, it does not account for any contributions from specular surface reflections, secondary light sources (other

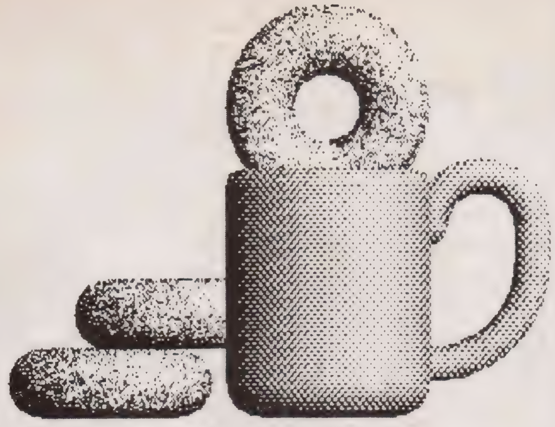


Figure 1

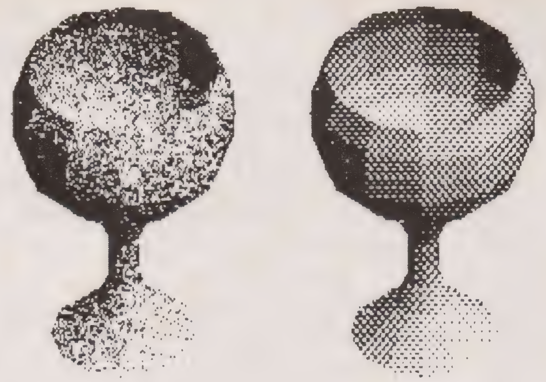


Figure 2

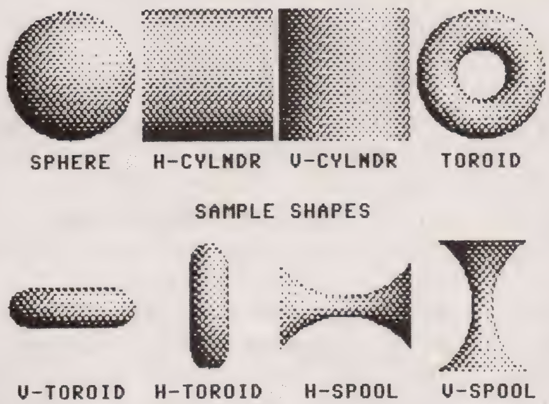


Figure 3

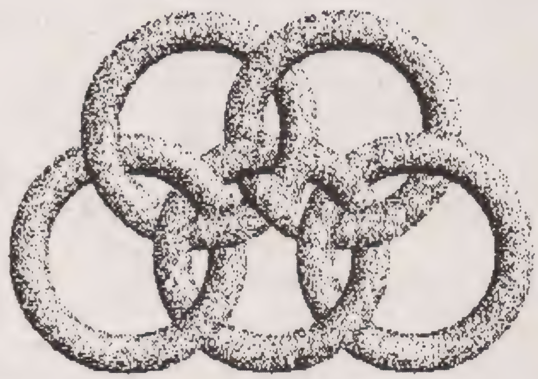


Figure 4

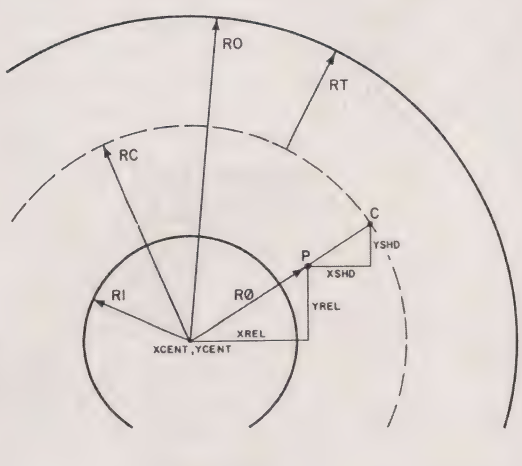


Figure 5

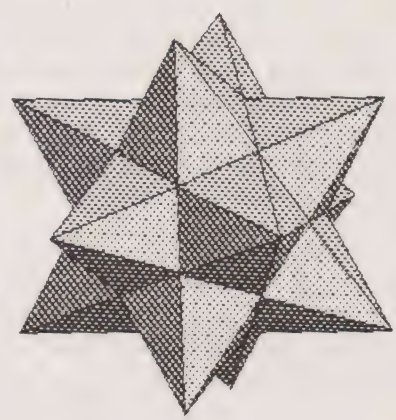


Figure 6

than the "opposite face" light) or ambient (nondirectional) light. These additional subtleties are not difficult to implement, but for the level of detail and dynamic range (number of gray levels) in our halftone display, they really aren't worth the effort.

Object Geometry

Calculating the surface normal vectors for points on an arbitrary surface can be difficult, but this is why we have restricted ourselves to objects that can be made up of combinations of the eight simple shapes shown in Figure 3 (page 55). All the basic shapes have at least fourfold symmetry; that is, once we have found the surface normal for a point (X, Y) in one quadrant of the object, we can easily find the surface normals for the other three quadrants by complementing appropriate components. Also, by restricting ourselves to shapes with relatively simple surfaces, we can determine the required surface normal vectors without resorting to trigonometric functions, using instead some simple geometric considerations.

Another time-saver is to choose an illumination reference vector to best take advantage of symmetries in the objects. Unfortunately, the "best" choice for computational purposes—a light source directly behind the viewer—gives a rather flat-looking image. Much more pleasing shading results from illumination coming from "over the shoulder." The slight assymmetry that results gives a stronger sense of depth to the objects.

The coordinate system describing objects on the screen has its X axis increasing horizontally to the right, the Y axis increasing vertically upward, and the Z axis increasing out of the screen toward the viewer (a conventional right-handed coordinate system). The illumination reference vector used here has X, Y, and Z components of (1,1,2), representing light coming from over the right shoulder. The symmetry in X and Y makes for easier shade calculations and gives a similar looking scene if you rotate a hard-copy output 90 degrees to fit objects that are taller than they are wide into the display. Lighting in the "portrait" (as opposed to "landscape"—unrotated) mode then comes from over the left shoulder.

Drawing a Shaded Sphere

Spheres are particularly easy objects to deal with because a radial vector from the center to a point on the surface is in the same direction as a surface normal from that point. The only real problem we must deal with is one of normalization. We will *always* deal with coordinates relative to the "local center of curvature" of objects. For spheres, this is just their geometric center.

Consider a point on the surface that extends out of the screen from relative coordinates (X,Y). We find the corresponding Z value using the equation for a sphere:

Z = SQR(R*R − X*X − Y*Y)

This calculated Z then forms the third member of the surface normal vector. To calculate the cosine of the angle between this vector and the illumination reference vector, we must first normalize both to have unit length. The normal we have just found has, of course, a length of R, so we need only divide each component by the radius of the sphere. The illumination vector (1,1,2) has a length of SQR(6), so we normalize this vector to unit length by dividing each component by SQR(6).

Now that we have two vectors of unit length but different directions, we find the cosine of the angle between them by taking their inner product, which is nothing more than summing the products of the X, Y, and Z coordinate pairs—like so:

COSINE = (1*X + 1*Y + 2*Z)
/ (R*SQR(6))

To use integer arithmetic throughout, we don't really want brightness expressed as a fraction between 0 and 1, but rather scaled to a range of integers from 0 to 63. We choose a maximum shade value of 63 to match the 64 gray levels that can be approximated by an 8×8 threshold matrix (described earlier in the "Graphics Utilities" section). We scale the pseudo-random bytes (used in the RSHADE routine) to the same range simply by shifting them right twice. So, for a surface point (X, Y, Z) on a sphere of radius R, the appropriate SHADE integer (0-63) is:

SHADE = 26*(X + Y + 2*Z)/R

The factor 26 properly accounts for the SQR (6) in the denominator of the previous equation.

As stated earlier, all the primitive shapes considered here have at least fourfold symmetry. We need compute the Z component only once for each ± X and ± Y quartet of points (relative to the geometric center of the object). The sphere and top-view toroid actually have n-fold symmetry, which the Cartesian grid of screen pixels reduces to eightfold symmetry. That is, in addition to changing signs of X and Y, we can exchange the X and Y coordinates to find another surface point with the same Z value.

Shape-Drawing Routines

Listing Four (page 68) is a set of routines for drawing the eight simple shapes shown in Figure 5. The routines resemble some form of compiled BASIC in that they use no special bit manipulations or unusual addressing methods, but just have appropriately ordered calls to the lower-level routines set up earlier. Comment statements precede each shape-drawing routine, giving an equivalent BASIC routine. This seems to be the easiest way to explain each routine because most readers are familiar with BASIC program methods. I thus give special comment to only a few of the routines here.

GETVAL is a shade-normalization routine that also checks for normal or backlit illumination via the flag BAKLIT. The byte pair at TONE contains the inner (dot) product of the illumination vector with the local surface normal vector. GETVAL then effectively does the division by SQR(6)*R and multiplication by 63 to put VALUE into the proper range, clipping at zero or taking the absolute value, as necessary.

PTPLOT does all the dirty work needed to plot shaded points for four-

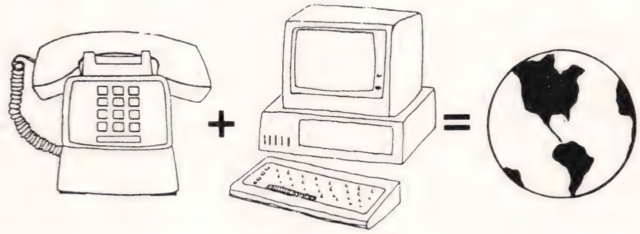
SOFTWARE DEVELOPERS

Add Touch Tone™ interface and Sound to your programs with

SoundTools™

Touch Tone data entry is now at your fingertips with this inexpensive programming package for the IBM PC, XT, AT and compatible personal computers.

SoundTools +



A whole new world of software applications!

Use Digital Pathways' SoundTools software (and Communicard™) to enhance your current applications and develop new ones that will:

— record and playback voice/sound
— provide remote data entry via Touch Tone
— perform automatic pulse and Touch Tone dialing
— detect call progress (ring, busy, voice, etc.)

SoundTools supports all of these languages... Interpreted and Compiled IBM BASICA and MS BASIC, MS PASCAL, C and Assembler.

List price only $449

Includes SoundTools software, Communicard, modular telephone cord and user's manual. *Ask about 21 day trial offer when placing your order.*

Communicard is a ½-size card that provides complete telephone interface, microphone and auxiliary output and Touch Tone decoding.

Requirements: PC/MS-DOS 2.0, 2.1, 3.0 or 3.1; IBM PC, XT, AT or compatible; 192K memory; DMA channel 1.

You too, can put your software on a SOUND FOOTING...

SoundWare™
SERIES OF SOFTWARE

DIGITAL PATHWAYS, INC.
1060 EAST MEADOW CIRCLE, PALO ALTO, CA 94303, 415-493-5544

Circle **no. 52** on reader service card.

fold symmetrical objects. Provision is also made here for clipping the object at independent levels up, down, left, and right of the object's center. The clipping feature is needed to blend various primitive objects into more complex ones with smooth transitions at the seams. Clipping also allows us to create some intricate "weaving" effects by redrawing portions of overlapping shapes as in Figure 4, "Linked Toroids," (page 55).

The flag NOROT determines whether the X and Y coordinates will be exchanged. This is useful both for drawing the eightfold symmetrical objects (top-view toroid and sphere) and for rotating the fourfold symmetrical objects 90 degrees so that a single drawing routine can produce two orientations of an object. The calculations are basically those for the sphere, but we'll see later that we can put other objects, such as toroids, into forms that look like spheres (at least locally).

GETZ calculates the effective Z coordinate for a spherelike surface, given the X and Y coordinates relative to the local center of curvature. Note that this routine needs the SQRT function, and because it is in general called for each quartet of points plotted, our fast SQRT makes a big difference in execution speed.

SPHERE follows the previously outlined algorithm and takes full advantage of the available symmetry. The radius is POKEd into RADIUS, and clipping distances relative to the sphere center are POKEd into CLIPL (left), CLIPR (right), CLIPU (up), and CLIPD (down).

CYLNDR draws a cylinder with sizes POKEd into RADIUS and HLEN (half-length). The flag NOROT determines whether the cylinder will be drawn with a horizontal or a vertical axis. The routine is simple, using PTPLOT for the main shade-calculation tasks. Actually, we could write a much faster routine to take advantage of the fact that we can use the same shade value for all points along lines parallel to the cylinder axis. This would require considerably more space, however.

The peculiar manipulations of relative coordinates for plotting toroids in various orientations are easier to explain with the help of the diagram in Figure 5 (appropriate for a top-view toroid) (page 55).

Consider a point P at coordinates (XREL, YREL) relative to the center of the object. We can take the local center of curvature of the toroid to be at the point C, which lies on the intersection of the line passing through the center of the toroid and a point beneath P (where Z=0) with a circle of radius RC (the "average" radius of the toroid is RC=(RO+RI) / 2). We determine the surface normal by the relative X, Y, and Z displacements from the point C.

The byte R0 is used for temporary storage of the radial distance from the toroid center to the point beneath P (where Z=0):

```
R0 = SQR(XREL*XREL
   + YREL*YREL)
```

We can determine the relative X and Y displacements, XSHD and YSHD, respectively, of P from C easily by using similar triangles:

```
XSHD = XREL*(1 – RC/R0)
YSHD = YREL*(1 – RC/R0)
```

We find the Z coordinate via the Pythagorean theorem, using the radius of the "ring" portion, RT, as the hypotenuse. RT is then also the length of the normal vector and is the value POKEd into RADIUS for normalization in the shade calculations.

The routine TOROID follows this algorithm, taking advantage of the eightfold symmetry in the object. EDGTOR and SPOOL use similar center of curvature coordinates to draw other orientations of a toroid. I have given the BASIC equivalents but leave it to readers to draw the appropriate geometric constructions if they desire more details.

Using the Shape-Drawing Routines—Interface to BASIC

The last module, completing our graphics package, is a convenient interface to BASIC. Listing Five (page 76) is a collection of routines to pass parameters easily to the graphics routines from the BASIC programs that control image generation. These routines are specific to the Commodore 64 in that they use some of the routines in the BASIC ROM to interpret statements in a BASIC program or entered in the direct command mode.

Graphics commands are given in the form:

```
SYS(FNCTN),PARAM1,PARAM2,
   [OPTIONAL PARAMETERS]
```

where FNCTN is the address of the graphic function you desire (or a variable that has been assigned the address value) and PARAM1 and PARAM2 are parameters (such as the center coordinates of an object) that the function requires, followed by any optional parameters.

The parameters can be either literal numerics or expressions that are evaluated for the desired values. We do not have to put the function address in parentheses, but it helps to make the program more readable. Separating parameters by commas is required. If you desire an optional parameter (such as specifying a new radius for sphere drawing), you add it by following the last required parameter by a comma and then the optional value or expression.

The use of optional parameters is more easily explained with the sequence of BASIC statements found in Figure 8 (page 59):

For all the shape-drawing functions, you must specify the X and Y center coordinates, but shape sizes are optional parameters. The last values specified become the new default sizes, allowing you to draw copies of similar objects (such as a bunch of grapes) just by setting the new center coordinates. All shapes but the sphere take two size parameters, and you must specify both when you desire a change (even when only one parameter takes on a new value). All size parameters (radii and cylinder half-lengths) must be less than 256, not a severe limitation considering the screen is only 200 pixels high ("240" with scaling).

The commands to clear the bit map and initialize the color map each take a single optional parameter, but the default values remain unchanged. Unless you specify alternate bytes, the bit

map is filled with 0s (cleared) and the color map is filled with 1s (for black dots on a white background). The most useful alternate byte with which to fill the bit map is 255, setting the entire background. Because the shading process clears, as well as sets, appropriate points, objects still appear normal but against a black background (particularly effective when you use the "back light" style).

The byte used to fill the color map is made up of background and foreground nibbles. To initialize the color map for a different color combination than black on white, use:

```
SYS(52001),16*DC+BC
```

where DC is the dot-color number and BC is the background-color number (as given in any reference to programming the Commodore 64).

The addresses for all the graphic functions and the parameters they take, along with locations to be POKEd with clipping values and flags for various drawing styles, are summarized in Table 3 (page 60). The best way to see how these shape-drawing routines are used is to examine the demonstration programs in Listing Six (page 78), "SHAPES DEMO," and Listing Seven (page 80), "STELLATION." All the different style options are exercized there. It is useful to save an abbreviated version of SHAPES DEMO, consisting of just the lines up to 340 and the subroutines following line 1620. This skeleton program can then form a base, providing all the POKE and SYS locations you need and to which you can add your own image-generation control program.

```
 10  REM ORDERED DITHER ARRAY—RECURSIVE FILL
 20  I=0:J=0:K=1:N=0:P=3
 30  REM 'P' IS ORDER OF ARRAY, SIZE IS (2 ^ P) x (2 ^ P)
 40  P=2 ^ P:DIM A(P - 1,P - 1)
 50  GOSUB 100
 60  END
100  IF K=P THEN A(I,J)=N:N=N+1:K=K/2:RETURN
110  K=2*K:GOSUB 100
120  I=I+K:J=J+K:K=2*K:GOSUB 100
130  I=I -K:K=2×K:GOSUB 100
140  I=I+K:J=J - K:K=2*K:GOSUB 100
150  I=I - K:K=K/2:RETURN
```

Figure 7

```
...  (SET UP GRAPHICS, STYLES, ETC)
200  SP=52119:REM ADDRESS OF SPHERE FUNCTION
210  SYS(SP),80,75,30:REM DRAW A SPHERE OF RADIUS 30 AT X=80, Y=75
220  SYS(SP),300,50 :REM DRAW ANOTHER SPHERE AT X=300, y=50
230  REM SINCE NO RADIUS IS SPECIFIED, LAST VALUE IS USED AS DEFAULT
240  SYS(SP),200,150,40:REM NEW SPHERE RADIUS (40) BECOMES DEFAULT
```

Figure 8

Auxiliary Programs

SHAPES DEMO and STELLATION make use of two auxiliary programs that, while not strictly needed to produce graphic images, provide utilities to both enhance the image itself and speed the drawing process. Listing Eight (page 80) is a routine for sorting an integer array indirectly through a key array (to determine drawing order for facets in a polygon mesh quickly). Listing Nine (page 82) allows you to add text to graphic

images in a variety of styles.

The code for these routines is short enough to be tucked in behind Commodore's DOS Wedge program so that, again, you don't lose any BASIC program space but maintain full compatibility with the wedge. If you are willing to give up the convenience of the wedge, however, you can easily relocate these routines to the end of the "interface" subprogram to keep the graphics package in one piece.

The KEYSORT routine works with two 1-dimensional integer arrays of the same size. One is filled with some "priority" parameter (such as the "average Z" coordinates for the facets in a polygon mesh), and the other becomes a "key" array whose elements index those in the "priority" array in increasing order.

To use the routine, POKE the maximum element index of the arrays into location 140, the address of the first (0th) element of the key array into locations 251 (low byte) and 252 (high byte), and the base address of the priority array into 253 and 254. You can find these array base addresses easily because the Commodore 64 places the address of the "current BASIC variable" into locations 71 and 72. Setting the 0th ele-

BIT-MAP SCREEN		: 40960-48959 ($A000-$BF3F)
COLOR MAP		: 33792-34791 ($8400-$87E7)
CLIPPING BOUNDS		(relative to object center):
893	($037D)	: LEFT BOUND
894	($037E)	: RIGHT BOUND
895	($037F)	: UPPER BOUND
896	($0380)	: LOWER BOUND
53280	($D020)	: BORDER COLOR [0-15], POKE with 1 to match white screen
STYLE FLAGS:		
838	($0346)	: SHADE STYLE [0=random, 1=halftone]
839	($0347)	: SCALE FLAG [0=normal (1:1), 1=scaled (3:4)]
868	($0364)	: EDGES FLAG [0=normal, 1=add lines to facet edges]
871	($0367)	: EDGE/LINE MODE [0=draw, 1=erase]
898	($0382)	: LIGHTING STYLE [0=normal single source, 1=backlit]
FUNCTION LOCATIONS—CALL WITH "SYS(FNCTN), PARAMETERS." OPTIONAL PARAMETERS IN SQUARE BRACKETS, DEFAULT VALUES ARE USED IF NOT SPECIFIED		
49378	($C0E2)	: SWITCH TO GRAPHICS MODE
49411	($C103)	: RETURN TO TEXT SCREEN
51979	($CB0B)	: CLEAR[,CLEAR BYTE] FILL BIT MAP WITH CLEAR BYTE [DEFAULT = 0]
52001	($CB21)	: COLOR[,COLOR BYTE] FILL COLOR MAP WITH COLOR BYTE
52023	($CB37)	: PLOT,X,Y SET POINT AT (X,Y)
52026	($CB3A)	: UNPLOT,X,Y CLEAR POINT AT (X,Y)
52049	($CB51)	: LINE,X1,Y1,X2,Y2 DRAW A LINE BETWEEN (X1,Y1) AND (X2,Y2)
52052	($CB54)	: FACET,X1,Y1,X2,Y2,X3,Y3,VA DRAW A SHADED TRIANGULAR FACET BETWEEN COORDINATE PAIRS (X1,Y1), (X2,Y2) AND (X3,Y3) USING "VA" SHADE VALUE (0:BLACK TO 64:WHITE)
CURVED SURFACES:		
52119	($CB97)	: SPHERE,X,Y[,R] DRAW A SPHERE CENTERED AT (X,Y) WITH RADIUS R
52141	($CBAD)	: TOROID,X,Y[,RI,RO] DRAW A TOP-VIEW TOROID WITH INNER RADIUS RI AND OUTER RADIUS RO
52150	($CBB6)	: VCYL,X,Y[,R,HL] DRAW A CYLINDER WITH AXIS VERTICAL, RADIUS R AND HALF-LENGTH HL
52153	($CBB9)	: HCYL,X,Y[,R,HL] DRAW A CYLINDER WITH AXIS HORIZONTAL
52186	($CBDA)	: VTOR,X,Y[,RI,RO] DRAW AN EDGE-VIEW TOROID WITH AXIS VERTICAL
52189	($CBDD)	: HTOR,X,Y[,RI,RO] DRAW AN EDGE-VEIW TOROID WITH AXIS HORIZONTAL
52203	($CBEB)	: VSPOOL,X,Y[,RI,RO] DRAW AN INSIDE-VIEW TOROID ("SPOOL") WITH AXIS VERTICAL
52206	($CBEE)	: HSPOOL,X,Y[,RI,RO] DRAW AN INSIDE-VIEW TOROID ("SPOOL") WITH AXIS HORIZONTAL

Table 3
Graphics Addresses

ment of an array equal to itself merely establishes it as the current variable. Then you can transfer the contents of 71 and 72 to the proper pointer at 251/252 or 253/254. Be careful to use literal numerics for "251" and so on, because you don't want to change the current variable. After setting up the pointers, SYS to the start of the KEYSORT routine. Another caution to observe if you use KEYSORT for other applications is *not* to create any new variables between finding the array base addresses and invoking KEYSORT, because arrays will be pushed up in memory to keep them at the end of BASIC's variable storage.

KEYSORT is set up to handle arrays with up to 256 elements each. If needed, you can sort larger arrays by breaking them into smaller arrays, sorting with KEYSORT, and then collating the results. Merging a few already sorted arrays is a fairly simple operation that you can do in a single pass, extending the usefulness of this routine.

STELLATION uses KEYSORT (lines 840–880) to determine the drawing order for the facets of a small stellated dodecahedron (Figure 6, page 55). This is a polyhedron that is not strictly convex (some internal dihedral angles are greater than 180 degrees), causing some foreground facets to partially obscure those in the background. The Painter's algorithm is used to handle hidden surface removal. Each newly drawn facet completely overwrites the background (clearing, as well as setting, points) so that drawing facets from back to front gives a proper rendering of the object.

The stellated dodecahedron of Figure 6 usually has only 30 visible or partially visible facets. Sorting with BASIC would be satisfactory here, but more complex polygon mesh objects can easily contain hundreds of visible and partially visible facets. You can really appreciate KEYSORT in those cases. Although it uses the simple and inefficient bubble-sort algorithm, being implemented in machine code lets KEYSORT run circles around any BASIC sorting alternative.

The final routine, in Listing Nine, WRITE, allows you to add text to the graphic display. The primary reason for implementing this as a machine code routine is that the bit map for the graphic image is located in the shadow RAM beneath the BASIC ROM (again saving useful BASIC program memory space). We can transfer bit images from the character ROM to the bit map just by POKEing values to the bit map, but we gain more flexibility if we first read the bit-map bytes to exclusive OR them, AND them, and so on with character image. Reading the shadow RAM requires that the BASIC ROM be switched out (through the bank switching used in the Commodore 64), something a BASIC program cannot do (and hope to continue execution).

The subroutine following line 1780 in SHAPES DEMO illustrates how to print text in a variety of styles on the bit-map image. Although the program actually uses only one style, the subroutine allows for five possibilities. Normal ("black" on "white") characters and reverse characters can overwrite the background, black characters can be ORed with the background, white characters ANDed with the background, or black characters exclusive ORed with the background. The last possibility gives characters that "change phase" across black/white edges in the image.

Conclusion

The graphics package described here we hope presents some new approaches to computer-generated images on small systems. At the same time, I'm sure I've missed some obvious tricks that would speed up the programs or reduce their size. I will be interested to see responses to this article and shortcuts or enhancements that might be added. Although the price/performance ratio of small graphics system continues to improve (especially with the development of custom LSI graphics chips), there are yet many unexplored software approaches to the problem. **DDJ**

Drawing on the C-64 **(Text begins on page 50)**

Listing One

```
0000                ; INTEGER ARITHMETIC ROUTINES
0000                ;
0000                ; RICHARD L. RYLANDER    8/12/84
0000                ;
0000                ; REVISED 10/29/84 TO ADD FULL DOUBLE
0000                ; PRECISION ARGUMENTS IN DIVIDE ROUTINE
0000                ;
0000                ;****************************************
0000                ;
0000                ; USE PAGE ZERO LOCATIONS WHERE POSSIBLE FOR
0000                ; ITERATIVE PROCEDURE WORK SPACE
0000                ;
0000                MLPCND =$AC             ; MULTIPLICAND
0000                MLPLER =$AD             ; MULTIPLIER
0000                PROD   =$AE             ; PRODUCT
0000                ;
0000                DVDND  =$FD             ; DIVIDEND/QUOTIENT
0000                DVSOR  =$FB             ; DIVISOR
0000                RMNDR  =$B4             ; REMAINDER
0000                ;
0000                RADCND =$AC             ; RADICAND
0000                ROOT   =$033C           ; SQUARE ROOT
0000                ;
0000                TEMP   =$FB
0000                ;
0000                ; SET UP SEED VALUES FOR PSEUDO-RANDOM NUMBERS
0000                       *=$C000
C000  FF            RNDM   .BYTE $FF,$55
C001  55
C002  00            RTEMP  .BYTE $00,$00
C003  00
C004                ;
C004                ;
C004                ;****************************************
C004                ;
C004                ; MULTIPLY SINGLE PRECISION MULTIPLICAND
C004                ; BY SINGLE PRECISION MULTIPLIER GIVING
C004                ; DOUBLE PRECISION PRODUCT (ENTER AT "MULT")
C004                ;
C004                ; SPECIAL CASE: ENTER AT "SQUARE" TO FIND
C004                ; SQUARE OF SIGNED 8-BIT NUMBER
C004                ;
C004  A5 AC         SQUARE LDA MLPCND       ; ENTRY TO SQUARE
C006  10 07                BPL POSITV       ; USE ABSOLUTE VALUE
C008  38                   SEC              ; NEGATE IF NEEDED
C009  A9 00                LDA #$00
C00B  E5 AC                SBC MLPCND
C00D  85 AC                STA MLPCND
C00F  85 AD         POSITV STA MLPLER
C011  A9 00         MULT   LDA #$00         ; ENTRY TO MULTIPLY
C013  A2 08                LDX #$08
C015  46 AD         MLOOP  LSR MLPLER
C017  90 03                BCC NOADD
C019  18                   CLC
C01A  65 AC                ADC MLPCND
C01C  6A            NOADD  ROR A
C01D  66 AE                ROR PROD
C01F  CA                   DEX
C020  D0 F3                BNE MLOOP
C022  85 AF                STA PROD+1
C024  60                   RTS
C025                ;
C025                ;****************************************
```

(Continued on next page)

Listing One

```
C025                ;
C025                ; DIVIDE DOUBLE PRECISION DIVIDEND
C025                ; BY DOUBLE PRECISION DIVISOR GIVING
C025                ; DOUBLE PRECISION QUOTIENT
C025                ;
C025                ; DIVIDEND IS REPLACED BY QUOTIENT
C025                ; IN THE PROCESS
C025                ;
C025                ; QUOTIENT IS ROUNDED TO NEAREST INTEGER
C025                ;
C025  A9 00         DIVIDE LDA #$00
C027  85 B4                STA RMNDR
C029  85 B5                STA RMNDR+1
C02B  A2 10                LDX #$10
C02D  26 FD         DLOOP  ROL DVDND
C02F  26 FE                ROL DVDND+1
C031  26 B4                ROL RMNDR
C033  26 B5                ROL RMNDR+1
C035  38                   SEC
C036  A5 B4                LDA RMNDR
C038  E5 FB                SBC DVSOR
C03A  A8                   TAY
C03B  A5 B5                LDA RMNDR+1
C03D  E5 FC                SBC DVSOR+1
C03F  90 04                BCC DECCNT
C041  84 B4                STY RMNDR
C043  85 B5                STA RMNDR+1
C045  CA            DECCNT DEX
C046  D0 E5                BNE DLOOP
C048  26 FD                ROL DVDND       ; CHECK IF REMAINDER
C04A  26 FE                ROL DVDND+1     ; IS >= 1/2 OF DIVIDEND
C04C  06 B4                ASL RMNDR       ; FOR ROUNDING
C04E  26 B5                ROL RMNDR+1
C050  B0 0B                BCS ROUND
C052  38                   SEC
C053  A5 FB                LDA DVSOR
C055  E5 B4                SBC RMNDR
C057  A5 FC                LDA DVSOR+1
C059  E5 B5                SBC RMNDR+1
C05B  B0 06                BCS NOCHNG
C05D  E6 FD         ROUND  INC DVDND
C05F  D0 02                BNE NOCHNG
C061  E6 FE                INC DVDND+1
C063  60            NOCHNG RTS
C064                ;
C064                ;****************************************
C064                ;
C064                ; TAKE INTEGER SQUARE ROOT OF A
C064                ; DOUBLE PRECISION RADICAND GIVING
C064                ; SINGLE PRECISION ROOT ( <= REAL ROOT )
C064                ;
C064  A2 08         SQRT   LDX #$08
C066  A9 00                LDA #$00
C068  8D 3C 03             STA ROOT
C06B  8D 3D 03             STA ROOT+1
C06E  85 FB                STA TEMP
C070  85 FC                STA TEMP+1
C072  0E 3C 03      SQRT1  ASL ROOT
C075  2E 3D 03             ROL ROOT+1
C078  EE 3C 03             INC ROOT        ; ASSUME CURRENT LSB OF
C07B  D0 03                BNE NEXT1       ; ROOT WILL BE 1
C07D  EE 3D 03             INC ROOT+1
C080  06 AC         NEXT1  ASL RADCND      ; SHIFT RADICAND LEFT
C082  26 AD                ROL RADCND+1    ; TWICE INTO TEMP
C084  26 FB                ROL TEMP
C086  26 FC                ROL TEMP+1
C088  06 AC                ASL RADCND
C08A  26 AD                ROL RADCND+1
C08C  26 FB                ROL TEMP
C08E  26 FC                ROL TEMP+1
C090  38                   SEC             ; SUBTRACT ROOT ESTIMATE
C091  A5 FB                LDA TEMP        ; FROM TEMP
C093  ED 3C 03             SBC ROOT
C096  A8                   TAY
C097  A5 FC                LDA TEMP+1
C099  ED 3D 03             SBC ROOT+1
C09C  90 12                BCC RESTOR
C09E  85 FC                STA TEMP+1      ; SUBTRACTION OK
C0A0  84 FB                STY TEMP
C0A2  EE 3C 03             INC ROOT
C0A5  D0 03                BNE NEXT2
C0A7  EE 3D 03             INC ROOT+1
C0AA  CA            NEXT2  DEX
C0AB  D0 C5                BNE SQRT1
C0AD  4C C1 C0             JMP FINI
C0B0  38            RESTOR SEC             ; IGNORE SUBTRACTION
C0B1  AD 3C 03             LDA ROOT        ; AND RESET LSB OF ROOT
C0B4  E9 01                SBC #$01
C0B6  8D 3C 03             STA ROOT
C0B9  B0 03                BCS NEXT3
C0BB  CE 3D 03             DEC ROOT+1
C0BE  CA            NEXT3  DEX
C0BF  D0 B1                BNE SQRT1
C0C1  6E 3D 03      FINI   ROR ROOT+1      ; FINAL /2 TO NORMALIZE
C0C4  6E 3C 03             ROR ROOT
C0C7  60                   RTS
C0C8                ;
C0C8                ;****************************************
C0C8                ;
C0C8                ; GENERATE PSEUDO-RANDOM BYTES
C0C8                ; EXIT WITH P-R BYTE IN ACCUM.
C0C8                ;
C0C8  AD 00 C0      RANDOM LDA RNDM
C0CB  8D 02 C0             STA RTEMP
C0CE  4D 01 C0             EOR RNDM+1
C0D1  2E 03 C0             ROL RTEMP+1     ; RTEMP+1 PRESERVES
C0D4  6A                   ROR A           ; CARRY BIT FOR CYCLING
C0D5  6E 03 C0             ROR RTEMP+1     ; RANDOM NUMBERS
C0D8  8D 00 C0             STA RNDM
C0DB  AD 02 C0             LDA RTEMP
C0DE  8D 01 C0             STA RNDM+1
C0E1  60                   RTS
C0E2                .END
ERRORS = 00000

ERRORS = 00000

SYMBOL TABLE

SYMBOL VALUE
 DECCNT   C045   DIVIDE   C025   DLOOP    C02D   DVDND    00FD
 DVSOR    00FB   FINI     C0C1   MLOOP    C015   MLPCND   00AC
 MLPLER   00AD   MULT     C011   NEXT1    C080   NEXT2    C0AA
 NEXT3    C0BE   NOADD    C01C   NOCHNG   C063   POSITV   C00F
 PROD     00AE   RADCND   00AC   RANDOM   C0C8   RESTOR   C0B0
 RMNDR    00B4   RNDM     C000   ROOT     033C   ROUND    C05D
 RTEMP    C002   SQRT     C064   SQRT1    C072   SQUARE   C004
 TEMP     00FB

END OF ASSEMBLY
```

End Listing One

Listing Two

```
0000                ; GRAPHICS UTILITIES
0000                ;
0000                ; RICHARD L. RYLANDER   11/4/84
0000                ;
0000                ; LOAD ARITHMETIC UTILITIES FIRST
0000                ;
0000                RAM=$033E
0000                ORIGIN=$C0E2
0000                ;
0000                MLPCND=$AC               ; MULTIPLICAND (S)
0000                MLPLER=$AD               ; MULTIPLIER (S)
0000                PROD=$AE                 ; PRODUCT (D)
0000                MULT=$C011               ; CALL FOR MULTIPLY
0000                ;
0000                RNDM=$C000               ; RANDOM NUMBER
0000                RANDOM=$C0C8             ; CALL FOR RANDOM
0000                ; NOTE - A CALL TO 'RANDOM' LEAVES A RANDOM BYTE
0000                ; IN THE ACCUMULATOR
0000                ;
0000                       *=RAM
033E                PLTFLG *=*+1             ; PLOT/UNPLOT FLAG
033F                XPLT   *=*+2             ; ABSOLUTE PLOT COORD
0341                YPLT   *=*+1             ; ABSOLUTE PLOT COORD
0342                VIC1   *=*+1             ; REGISTER STORAGE
0343                VIC2   *=*+1             ; REGISTER STORAGE
0344                VALUE  *=*+2             ; FINAL NORMALIZED SHADE VALUE
0346                HTORRN *=*+1             ; SHADE FLAG, 1=HALFTONE
0347                NOSCAL *=*+1             ; SCALE FLAG, 1=NO SCALE
0348                TEMP   *=*+2             ; TEMPORARY STORAGE
034A                ;
034A                       *=ORIGIN
C0E2                ;
C0E2                ;****************************************
C0E2                ;
C0E2                ; TURN ON BIT MAP GRAPHICS MODE,
C0E2                ; SAVING REGISTER VALUES FOR
C0E2                ; RETURN TO TEXT MODE LATER.
C0E2                ;
C0E2  AD 11 D0      GRFON  LDA $D011
C0E5  09 20                ORA #$20
C0E7  8D 11 D0             STA $D011
C0EA  AD 00 DD             LDA $DD00
C0ED  8D 42 03             STA VIC1
C0F0  29 FC                AND #$FC
C0F2  09 01                ORA #$01
C0F4  8D 00 DD             STA $DD00
C0F7  AD 18 D0             LDA $D018
C0FA  8D 43 03             STA VIC2
C0FD  A9 19                LDA #$19
C0FF  8D 18 D0             STA $D018
C102  60                   RTS
C103                ;
C103                ;****************************************
C103                ;
C103                ; RETURN TO TEXT SCREEN
C103                ;
C103  AD 11 D0      GRFOFF LDA $D011
C106  29 DF                AND #$DF
C108  8D 11 D0             STA $D011
C10B  AD 42 03             LDA VIC1
C10E  8D 00 DD             STA $DD00
C111  AD 43 03             LDA VIC2
C114  8D 18 D0             STA $D018
C117  60                   RTS
C118                ;
C118                ;****************************************
C118                ;
C118                ; FILL COLOR MAP FOR BLACK DOTS ON WHITE
C118                ;
C118  A9 01         COLOR  LDA #$01        ; POKE NEW COLORS HERE
C11A  A2 00                LDX #0
C11C  9D 00 84      COL1   STA $8400,X
C11F  9D 00 85             STA $8500,X
C122  9D 00 86             STA $8600,X
C125  9D 00 87             STA $8700,X
C128  CA                   DEX
C129  D0 F1                BNE COL1
C12B  60                   RTS
C12C                ;
C12C                ;****************************************
C12C                ;
C12C                ; CLEAR HI-RES GRAPHICS SCREEN
C12C                ;
C12C  A9 A0         CLEAR  LDA #$A0
C12E  85 FC                STA $FC
C130  A0 00                LDY #0
C132  84 FB                STY $FB
C134  A9 00                LDA #0          ; CLEAR BYTE
C136  A2 20                LDX #$20
C138  91 FB         CLRLP  STA ($FB),Y
C13A  C8                   INY
C13B  D0 FB                BNE CLRLP
C13D  E6 FC                INC $FC
C13F  CA                   DEX
C140  D0 F6                BNE CLRLP
C142  60                   RTS
C143                ;
C143                ;****************************************
C143                ;
C143                ; PLOT AND UNPLOT POINTS ON HI-RES GRAPHICS
C143                ; SCREEN. ABSOLUTE X AND Y SCREEN COORDINATES
C143                ; ARE POKED INTO XPLT, XPLT+1, AND YPLT
C143                ;
C143  A9 00         PLOT   LDA #0
C145  2C                   .BYTE $2C
C146  A9 80         UNPLOT LDA #$80
C148  8D 3E 03             STA PLTFLG
C14B  A5 01                LDA $01         ; BASIC ROM OUT
C14D  29 FE                AND #$FE
C14F  85 01                STA $01
C151  38                   SEC             ; INVERT Y COORDINATE TO
C152  A9 C7                LDA #$C7        ; PUT ORIGIN IN LOWER LEFT
C154  ED 41 03             SBC YPLT        ; CORNER OF SCREEN
C157  AA                   TAX             ; (199.-YPLT)
C158  4A                   LSR A
C159  4A                   LSR A
C15A  4A                   LSR A
C15B  A8                   TAY
C15C  B9 AB C1             LDA TABLE1,Y
C15F  85 FB                STA $FB
C161  B9 C4 C1             LDA TABLE2,Y
C164  85 FC                STA $FC
```

(Continued on page 64)

Listing Two

```
00123  C166  8A                TXA
00124  C167  29 07             AND #$07
00125  C169  18                CLC
00126  C16A  65 FB             ADC $FB
00127  C16C  85 FB             STA $FB
00128  C16E  AD 3F 03          LDA XPLT
00129  C171  29 F8             AND #$F8
00130  C173  65 FB             ADC $FB
00131  C175  85 FB             STA $FB
00132  C177  AD 40 03          LDA XPLT+1
00133  C17A  65 FC             ADC $FC
00134  C17C  85 FC             STA $FC
00135  C17E  A9 A0             LDA #$A0
00136  C180  65 FC             ADC $FC
00137  C182  85 FC             STA $FC
00138  C184  AD 3F 03          LDA XPLT
00139  C187  29 07             AND #$07
00140  C189  49 07             EOR #$07
00141  C18B  AA                TAX
00142  C18C  A9 01             LDA #$01
00143  C18E  CA        PLOTLP  DEX
00144  C18F  30 03             BMI PLOT2
00145  C191  0A                ASL A
00146  C192  D0 FA             BNE PLOTLP
00147  C194  A0 00     PLOT2   LDY #0
00148  C196  2C 3E 03          BIT PLTFLG
00149  C199  10 05             BPL NOPLOT
00150  C19B  49 FF             EOR #$FF
00151  C19D  31 FB             AND ($FB),Y
00152  C19F  2C                .BYTE $2C
00153  C1A0  11 FB     NOPLOT  ORA ($FB),Y
00154  C1A2  91 FB             STA ($FB),Y
00155  C1A4  A5 01             LDA $01          ; BASIC ROM RESTORED
00156  C1A6  09 01             ORA #$01
00157  C1A8  85 01             STA $01
00158  C1AA  60                RTS
00159  C1AB            ;
00160  C1AB  00        TABLE1 .BYTE $00,$40,$80,$C0
00160  C1AC  40
00160  C1AD  80
00160  C1AE  C0
00161  C1AF  00               .BYTE $00,$40,$80,$C0
00161  C1B0  40
00161  C1B1  80
00161  C1B2  C0
00162  C1B3  00               .BYTE $00,$40,$80,$C0
00162  C1B4  40
00162  C1B5  80
00162  C1B6  C0
00163  C1B7  00               .BYTE $00,$40,$80,$C0
00163  C1B8  40
00163  C1B9  80
00163  C1BA  C0
00164  C1BB  00               .BYTE $00,$40,$80,$C0
00164  C1BC  40
00164  C1BD  80
00164  C1BE  C0
00165  C1BF  00               .BYTE $00,$40,$80,$C0,$00
00165  C1C0  40
00165  C1C1  80
00165  C1C2  C0
00165  C1C3  00
00166  C1C4            ;
00167  C1C4  00        TABLE2 .BYTE $00,$01,$02,$03
00167  C1C5  01
00167  C1C6  02
00167  C1C7  03
00168  C1C8  05               .BYTE $05,$06,$07,$08
00168  C1C9  06
00168  C1CA  07
00168  C1CB  08
00169  C1CC  0A               .BYTE $0A,$0B,$0C,$0D
00169  C1CD  0B
00169  C1CE  0C
00169  C1CF  0D
00170  C1D0  0F               .BYTE $0F,$10,$11,$12
00170  C1D1  10
00170  C1D2  11
00170  C1D3  12
00171  C1D4  14               .BYTE $14,$15,$16,$17
00171  C1D5  15
00171  C1D6  16
00171  C1D7  17
00172  C1D8  19               .BYTE $19,$1A,$1B,$1C,$1E
00172  C1D9  1A
00172  C1DA  1B
00172  C1DB  1C
00172  C1DC  1E
00173  C1DD            ;
00174  C1DD            ;****************************************
00175  C1DD            ;
00176  C1DD            ; SHADING BY HYBRID DITHER/DOT-GROWTH
00177  C1DD            ;
00178  C1DD  AD 3F 03  SHADE  LDA XPLT         ; USE BITS -----***
00179  C1E0  29 07            AND #$07         ; OF 'X' SCREEN COORD
00180  C1E2  8D 48 03         STA TEMP
00181  C1E5  AD 41 03         LDA YPLT         ; AND BITS -----***
00182  C1E8  29 07            AND #$07         ; OF 'Y' SCREEN COORD
00183  C1EA  0A               ASL A            ; SHIFTED INTO --***---
00184  C1EB  0A               ASL A            ; POSITION TO DETERMINE
00185  C1EC  0A               ASL A            ; 6-BIT OFFSET IN
00186  C1ED  0D 48 03         ORA TEMP         ; THRESHOLD TABLE
00187  C1F0  AA               TAX
00188  C1F1  BD 2F C2         LDA THRESH,X     ; SCREEN-POSITION-WEIGHTED
00189  C1F4  CD 44 03         CMP VALUE        ; THRESHOLD VALUE
00190  C1F7  10 03            BPL GREATR
00191  C1F9  4C 46 C1         JMP UNPLOT
00192  C1FC  4C 43 C1  GREATR JMP PLOT
00193  C1FF            ;
00194  C1FF            ;****************************************
00195  C1FF            ;
00196  C1FF            ; SHADING BY RANDOM HALFTONE
00197  C1FF            ;
00198  C1FF  20 C8 C0  RSHADE JSR RANDOM
00199  C202  4A               LSR A            ; REDUCE RANDOM BYTE
00200  C203  4A               LSR A            ; TO 6 BITS FOR SHADE
00201  C204  CD 44 03         CMP VALUE        ; VALUE COMPARISON
00202  C207  10 03            BPL MORE
00203  C209  4C 46 C1         JMP UNPLOT
00204  C20C  4C 43 C1  MORE   JMP PLOT
00205  C20F            ;
00206  C20F            ;****************************************
00207  C20F            ;
00208  C20F            ; PLOT A POINT WEIGHTED BY SHADING SCHEME
00209  C20F            ; AND SHADE VALUE
00210  C20F            ; CHECK 'NOSCAL' FLAG FOR SCALING OF Y COORD
00211  C20F            ; CHECK 'HTORRN' FLAG FOR TYPE OF SHADING
00212  C20F            ;
00213  C20F  AD 47 03  PLTSHD LDA NOSCAL
00214  C212  F0 10            BEQ NORM
00215  C214            ;
00216  C214            ; SCALE Y FROM 0-239 PSEUDO-COORDINATES
00217  C214            ; TO 0-199 TRUE SCREEN COORDINATES BY
00218  C214            ;     Y = (Y+1)*213/256
00219  C214            ;
00220  C214  AC 41 03  SCALE  LDY YPLT
00221  C217  C8               INY
00222  C218  84 AD            STY MLPLER
00223  C21A  A9 D5            LDA #$D5         ; 213.
00224  C21C  85 AC            STA MLPCND
00225  C21E  20 11 C0         JSR MULT         ; RETURN WITH HIGH BYTE
00226  C221  8D 41 03         STA YPLT         ; IN ACCUMULATOR
00227  C224  AD 46 03  NORM   LDA HTORRN
00228  C227  F0 03            BEQ RPLT
00229  C229  4C DD C1         JMP SHADE
00230  C22C  4C FF C1  RPLT   JMP RSHADE
00231  C22F            ;
00232  C22F            ;
00233  C22F            ;
00234  C22F  00        THRESH .BYTE $00,$08,$35,$3D
00234  C230  08
00234  C231  35
00234  C232  3D
00235  C233  02               .BYTE $02,$0A,$37,$3F
00235  C234  0A
00235  C235  37
00235  C236  3F
00236  C237  10               .BYTE $10,$18,$25,$2D
00236  C238  18
00236  C239  25
00236  C23A  2D
00237  C23B  12               .BYTE $12,$1A,$27,$2F
00237  C23C  1A
00237  C23D  27
00237  C23E  2F
00238  C23F  31               .BYTE $31,$39,$04,$0C
00238  C240  39
00238  C241  04
00238  C242  0C
00239  C243  33               .BYTE $33,$3B,$06,$0E
00239  C244  3B
00239  C245  06
00239  C246  0E
00240  C247  21               .BYTE $21,$29,$14,$1C
00240  C248  29
00240  C249  14
00240  C24A  1C
00241  C24B  23               .BYTE $23,$2B,$16,$1E
00241  C24C  2B
00241  C24D  16
00241  C24E  1E
00242  C24F  03               .BYTE $03,$0B,$36,$3E
00242  C250  0B
00242  C251  36
00242  C252  3E
00243  C253  01               .BYTE $01,$09,$34,$3C
00243  C254  09
00243  C255  34
00243  C256  3C
00244  C257  13               .BYTE $13,$1B,$26,$2E
00244  C258  1B
00244  C259  26
00244  C25A  2E
00245  C25B  11               .BYTE $11,$19,$24,$2C
00245  C25C  19
00245  C25D  24
00245  C25E  2C
00246  C25F  32               .BYTE $32,$3A,$07,$0F
00246  C260  3A
00246  C261  07
00246  C262  0F
00247  C263  30               .BYTE $30,$38,$05,$0D
00247  C264  38
00247  C265  05
00247  C266  0D
00248  C267  22               .BYTE $22,$2A,$17,$1F
00248  C268  2A
00248  C269  17
00248  C26A  1F
00249  C26B  20               .BYTE $20,$28,$15,$1D
00249  C26C  28
00249  C26D  15
00249  C26E  1D
00250  C26F                  .END

ERRORS = 00000

SYMBOL TABLE

SYMBOL VALUE
 CLEAR    C12C    CLRLP    C13B    COL1     C11C    COLOR    C118
 GREATR   C1FC    GRFOFF   C103    GRFON    C0E2    HTORRN   0346
 MLPCND   00AC    MLPLER   00AD    MORE     C20C    MULT     C011
 NOPLOT   C1A0    NORM     C224    NOSCAL   0347    ORIGIN   C0E2
 PLOT     C143    PLOT2    C194    PLOTLP   C18E    PLTFLG   033E
 PLTSHD   C20F    PROD     00AE    RAM      033E    RANDOM   C0C8
 RNDM     C000    RPLT     C22C    RSHADE   C1FF    SCALE    C214
 SHADE    C1DD    TABLE1   C1AB    TADLE2   C1C4    TEMP     0348
 THRESH   C22F    UNPLOT   C146    VALUE    0344    VIC1     0342
 VIC2     0343    XPLT     033F    YPLT     0341

END OF ASSEMBLY
```

End Listing Two

Listing Three

```
00001  0000            ; FACET - DRAW SHADED TRIANGULAR FACETS
00002  0000            ; AND STRAIGHT LINES.
00003  0000            ;
00004  0000            ; RICHARD L. RYLANDER   11/4/84
00005  0000            ;
00006  0000            ; LOAD "ARITH.HEX" AND "GRAPH.HEX"
00007  0000            ; BEFORE USING
00008  0000            ;
00009  0000            ORIGIN = $C26F
00010  0000            RAM    = $034A
00011  0000            ;
00012  0000            XPLT   = $033F
00013  0000            YPLT   = $0341
00014  0000            NORM   = $C224
00015  0000            NOSCAL = $0347
00016  0000            PLOT   = $C143
00017  0000            UNPLOT = $C146
```

(Continued on page 66)

Listing Three

```
0000                      ;
0000               MLPCND = $AC
0000               MLPLER = $AD
0000               PROD   = $AE
0000               MULT   = $C011
0000               ;
0000               DVDND  = $FD
0000               DVSOR  = $FB
0000               QUOT   = $FD
0000               DIVIDE = $C025
0000               ;
0000                      *=RAM
034A               ;
034A               XMIN   *=*+2
034C               YMIN   *=*+1
034D               XMID   *=*+2
034F               YMID   *=*+1
0350               XMAX   *=*+2
0352               YMAX   *=*+1
0353               YTOP   *=*+1
0354               YBOT   *=*+1
0355               YBASE  *=*+1
0356               DLTAX1 *=*+2
0358               DLTAX2 *=*+1
0359               DLTAX3 *=*+1
035A               DELTAX *=*+1
035B               DLTAY1 *=*+1
035C               DLTAY2 *=*+1
035D               DLTAY3 *=*+1
035E               DELTAY *=*+1
035F               XDIFF  *=*+1
0360               FLAG1  *=*+1
0361               FLAG2  *=*+1
0362               FLAG3  *=*+1
0363               FLAG   *=*+1
0364               EDGES  *=*+1
0365               ERROR  *=*+2
0367               MODE   *=*+1
0368               COUNT  *=*+2
036A               ;
036A               ;
036A                      *=ORIGIN
C26F               ;
C26F               ;*****************************************
C26F               ;
C26F               ; SCALE ALL Y COORDINATES FROM 0..239
C26F               ; PSUEDO-COORDINATE RANGE TO 0..199
C26F               ; TRUE SCREEN COORDINATE RANGE
C26F               ;
C26F  A0 06        SCALE  LDY #6
C271  A9 D5               LDA #$D5
C273  85 AC               STA MLPCND
C275  B9 4C 03     SCLP   LDA YMIN,Y
C278  85 AD               STA MLPLER
C27A  20 11 C0            JSR MULT
C27D  99 4C 03            STA YMIN,Y
C280  88                  DEY
C281  88                  DEY
C282  88                  DEY
C283  10 F0               BPL SCLP
C285  60                  RTS
C286               ;
C286               ;
C286               ;*****************************************
C286               ;
C286               ; EXCHANGE 'MIN' AND 'MID' COORDINATES
C286               ;
C286  A0 02        SWAP12 LDY #2
C288  B9 4A 03     LOOP1  LDA XMIN,Y
C28B  48                  PHA
C28C  B9 4D 03            LDA XMID,Y
C28F  99 4A 03            STA XMIN,Y
C292  68                  PLA
C293  99 4D 03            STA XMID,Y
C296  88                  DEY
C297  10 EF               BPL LOOP1
C299  60                  RTS
C29A               ;
C29A               ;*****************************************
C29A               ;
C29A               ; EXCHANGE 'MID' AND 'MAX' COORDINATES
C29A               ;
C29A  A0 02        SWAP23 LDY #2
C29C  B9 4D 03     LOOP2  LDA XMID,Y
C29F  48                  PHA
C2A0  B9 50 03            LDA XMAX,Y
C2A3  99 4D 03            STA XMID,Y
C2A6  68                  PLA
C2A7  99 50 03            STA XMAX,Y
C2AA  88                  DEY
C2AB  10 EF               BPL LOOP2
C2AD  60                  RTS
C2AE               ;
C2AE               ;*****************************************
C2AE               ;
C2AE               ; SORT COORDINATES ACCORDING TO X COMPONENTS
C2AE               ;
C2AE  A2 02        SORTX  LDX #2
C2B0  38           SORTLP SEC
C2B1  AD 4D 03            LDA XMID
C2B4  ED 4A 03            SBC XMIN
C2B7  AD 4E 03            LDA XMID+1
C2BA  ED 4B 03            SBC XMIN+1
C2BD  B0 03               BCS NOSWP1
C2BF  20 86 C2            JSR SWAP12
C2C2  CA           NOSWP1 DEX
C2C3  F0 15               BEQ SORTED
C2C5  38                  SEC
C2C6  AD 50 03            LDA XMAX
C2C9  ED 4D 03            SBC XMID
C2CC  AD 51 03            LDA XMAX+1
C2CF  ED 4E 03            SBC XMID+1
C2D2  B0 DC               BCS SORTLP
C2D4  20 9A C2            JSR SWAP23
C2D7  4C B0 C2            JMP SORTLP
C2DA  60           SORTED RTS
C2DB               ;
C2DB               ;*****************************************
C2DB               ;
C2DB               ; DRAW A LINE BETWEEN XMIN,YMIN AND XMID,YMID
C2DB               ; USING FAST DDA (DIGITAL DIFFERENTIAL ANALYZER)
C2DB               ; TECHNIQUE
C2DB               ;
C2DB  A9 02        LINE   LDA #2          ; ENSURE XMAX IS
C2DD  8D 51 03            STA XMAX+1      ; LARGEST BEFORE
C2E0  20 AE C2            JSR SORTX       ; ORDERING 'MIN' AND 'MID'
C2E3  AD 47 03            LDA NOSCAL
C2E6  F0 03               BEQ OUTLN
C2E8  20 6F C2            JSR SCALE
C2EB  20 6F C4     OUTLN  JSR FINDXY      ; ENTRY POINT TO
C2EE  AD 4A 03            LDA XMIN        ; OUTLINE FACETS
C2F1  8D 3F 03            STA XPLT
C2F4  AD 4B 03            LDA XMIN+1
C2F7  8D 40 03            STA XPLT+1
C2FA  AD 4C 03            LDA YMIN
C2FD  8D 41 03            STA YPLT
C300  AD 57 03            LDA DLTAX1+1    ; CHECK FOR DX>DY
C303  D0 7D               BNE STEPX
C305  38                  SEC
C306  AD 56 03            LDA DLTAX1
C309  ED 5B 03            SBC DLTAY1
C30C  B0 74               BCS STEPX
C30E  AD 5B 03     STEPY  LDA DLTAY1
C311  8D 65 03            STA ERROR
C314  8D 68 03            STA COUNT
C317  4E 65 03            LSR ERROR
C31A  38                  SEC
C31B  AD 56 03            LDA DLTAX1
C31E  ED 65 03            SBC ERROR
C321  8D 65 03            STA ERROR
C324  AD 57 03            LDA DLTAX1+1
C327  E9 00               SBC #0
C329  8D 66 03            STA ERROR+1
C32C  EE 68 03            INC COUNT
C32F  AD 67 03     LNLP1  LDA MODE        ; 0 = DRAW, 1 = ERASE
C332  D0 06               BNE ERASE1
C334  20 43 C1            JSR PLOT
C337  4C 3D C3            JMP SK1
C33A  20 46 C1     ERASE1 JSR UNPLOT
C33D  AD 60 03     SK1    LDA FLAG1       ; 0 = POSITIVE SLOPE
C340  D0 05               BNE NSLOPE
C342  EE 41 03            INC YPLT
C345  D0 03               BNE SK2         ; ALWAYS BRANCH
C347  CE 41 03     NSLOPE DEC YPLT
C34A  2C 66 03     SK2    BIT ERROR+1
C34D  30 1A               BMI SK3
C34F  EE 3F 03            INC XPLT
C352  D0 03               BNE NOINC1
C354  EE 40 03            INC XPLT+1
C357  38           NOINC1 SEC
C358  AD 65 03            LDA ERROR
C35B  ED 5B 03            SBC DLTAY1
C35E  8D 65 03            STA ERROR
C361  AD 66 03            LDA ERROR+1
C364  E9 00               SBC #0
C366  8D 66 03            STA ERROR+1
C369  18           SK3    CLC
C36A  AD 65 03            LDA ERROR
C36D  6D 56 03            ADC DLTAX1
C370  8D 65 03            STA ERROR
C373  AD 66 03            LDA ERROR+1
C376  6D 57 03            ADC DLTAX1+1
C379  8D 66 03            STA ERROR+1
C37C  CE 68 03            DEC COUNT
C37F  D0 AE               BNE LNLP1
C381  60                  RTS
C382               ;
C382  AD 56 03     STEPX  LDA DLTAX1
C385  8D 65 03            STA ERROR
C388  8D 68 03            STA COUNT
C38B  AD 57 03            LDA DLTAX1+1
C38E  8D 66 03            STA ERROR+1
C391  8D 69 03            STA COUNT+1
C394  4E 66 03            LSR ERROR+1
C397  6E 65 03            ROR ERROR
C39A  38                  SEC
C39B  AD 5B 03            LDA DLTAY1
C39E  ED 65 03            SBC ERROR
C3A1  8D 65 03            STA ERROR
C3A4  A9 00               LDA #0
C3A6  ED 66 03            SBC ERROR+1
C3A9  8D 66 03            STA ERROR+1
C3AC  AD 67 03     LNLP2  LDA MODE
C3AF  D0 06               BNE ERASE2
C3B1  20 43 C1            JSR PLOT
C3B4  4C BA C3            JMP SKP1
C3B7  20 46 C1     ERASE2 JSR UNPLOT
C3BA  EE 3F 03     SKP1   INC XPLT
C3BD  D0 03               BNE NOINC2
C3BF  EE 40 03            INC XPLT+1
C3C2  2C 66 03     NOINC2 BIT ERROR+1
C3C5  30 20               BMI SKP3
C3C7  AD 60 03            LDA FLAG1
C3CA  D0 05               BNE NGSLP
C3CC  EE 41 03            INC YPLT
C3CF  D0 03               BNE SKP2        ; ALWAYS BRANCH
C3D1  CE 41 03     NGSLP  DEC YPLT
C3D4  38           SKP2   SEC
C3D5  AD 65 03            LDA ERROR
C3D8  ED 56 03            SBC DLTAX1
C3DB  8D 65 03            STA ERROR
C3DE  AD 66 03            LDA ERROR+1
C3E1  ED 57 03            SBC DLTAX1+1
C3E4  8D 66 03            STA ERROR+1
C3E7  18           SKP3   CLC
C3E8  AD 65 03            LDA ERROR
C3EB  6D 5B 03            ADC DLTAY1
C3EE  8D 65 03            STA ERROR
C3F1  AD 66 03            LDA ERROR+1
C3F4  69 00               ADC #0
C3F6  8D 66 03            STA ERROR+1
C3F9  38                  SEC
C3FA  AD 68 03            LDA COUNT
C3FD  E9 01               SBC #1
C3FF  8D 68 03            STA COUNT
C402  B0 03               BCS TEST
C404  CE 69 03            DEC COUNT+1
C407  2C 69 03     TEST   BIT COUNT+1
C40A  10 A0               BPL LNLP2
C40C  60                  RTS
C40D               ;
C40D               ;*****************************************
C40D               ; DRAW A SHADED VERTICAL LINE AT
C40D               ; XPLT FROM YTOP TO YBOT
C40D               ;
C40D  38           VLINE  SEC             ; MAKE SURE YTOP>YBOT
C40E  AD 53 03            LDA YTOP
C411  ED 54 03            SBC YBOT
C414  B0 0E               BCS DRAW
C416  AD 53 03            LDA YTOP
C419  48                  PHA
C41A  AD 54 03            LDA YBOT
C41D  8D 53 03            STA YTOP
C420  68                  PLA
C421  8D 54 03            STA YBOT
```

(Continued on page 68)

Listing Three

```
C424  AD 53 03   DRAW   LDA YTOP
C427  8D 41 03          STA YPLT
C42A  20 24 C2          JSR NORM        ; PLOT A SHADE-WEIGHTED
C42D  AD 53 03          LDA YTOP        ; PIXEL CHECKING ONLY
C430  CD 54 03          CMP YBOT        ; FOR SHADE STYLE
C433  F0 06             BEQ DONE
C435  CE 53 03          DEC YTOP
C438  4C 24 C4          JMP DRAW
C43B  60         DONE   RTS
C43C                    ;
C43C                    ;****************************************
C43C                    ;
C43C                    ; FIND ENDPOINTS FOR VERTICAL LINES
C43C                    ; BETWEEN FACET EDGES
C43C                    ;
C43C  AD 5F 03   ENDPTS LDA XDIFF
C43F  85 AC             STA MLPCND
C441  AD 5E 03          LDA DELTAY
C444  85 AD             STA MLPLER
C446  20 11 C0          JSR MULT
C449  85 FE             STA DVDND+1
C44B  A5 AE             LDA PROD
C44D  85 FD             STA DVDND
C44F  A9 00             LDA #0
C451  85 FC             STA DVSOR+1
C453  AD 5A 03          LDA DELTAX
C456  85 FB             STA DVSOR
C458  20 25 C0          JSR DIVIDE
C45B  AD 63 03          LDA FLAG
C45E  D0 08             BNE NEGSLP
C460  18                CLC
C461  AD 55 03          LDA YBASE
C464  65 FD             ADC QUOT
C466  90 06             BCC SKIP2
C468  38         NEGSLP SEC
C469  AD 55 03          LDA YBASE
C46C  E5 FD             SBC QUOT
C46E  60         SKIP2  RTS
C46F                    ;
C46F                    ;****************************************
C46F                    ;
C46F                    ; FIND COORDINATE DIFFERENCES
C46F                    ;
C46F                    ; ALL "DELTA X" VALUES POSITIVE,
C46F                    ; SINGLE PRECISION (JUST LOWER BYTE)
C46F                    ;
C46F  38         FINDXY SEC
C470  AD 4D 03          LDA XMID
C473  ED 4A 03          SBC XMIN
C476  8D 56 03          STA DLTAX1
C479  AD 4E 03          LDA XMID+1
C47C  ED 4B 03          SBC XMIN+1
C47F  8D 57 03          STA DLTAX1+1
C482  38                SEC
C483  AD 50 03          LDA XMAX
C486  ED 4D 03          SBC XMID
C489  8D 58 03          STA DLTAX2
C48C  38                SEC
C48D  AD 50 03          LDA XMAX
C490  ED 4A 03          SBC XMIN
C493  8D 59 03          STA DLTAX3
C496                    ;
C496                    ; USE ABS(DELTA Y) VALUES,
C496                    ; FLAGS INDICATE SLOPE OF LIMIT LINES
C496                    ;
C496  A9 00             LDA #$00
C498  8D 60 03          STA FLAG1
C49B  8D 61 03          STA FLAG2
C49E  8D 62 03          STA FLAG3
C4A1  38                SEC
C4A2  AD 4F 03          LDA YMID
C4A5  ED 4C 03          SBC YMIN
C4A8  B0 09             BCS STORE1
C4AA  EE 60 03          INC FLAG1
C4AD  AD 4C 03          LDA YMIN
C4B0  ED 4F 03          SBC YMID
C4B3  8D 5B 03   STORE1 STA DLTAY1
C4B6  38                SEC
C4B7  AD 52 03          LDA YMAX
C4BA  ED 4F 03          SBC YMID
C4BD  B0 09             BCS STORE2
C4BF  EE 61 03          INC FLAG2
C4C2  AD 4F 03          LDA YMID
C4C5  ED 52 03          SBC YMAX
C4C8  8D 5C 03   STORE2 STA DLTAY2
C4CB  38                SEC
C4CC  AD 52 03          LDA YMAX
C4CF  ED 4C 03          SBC YMIN
C4D2  B0 09             BCS STORE3
C4D4  EE 62 03          INC FLAG3
C4D7  AD 4C 03          LDA YMIN
C4DA  ED 52 03          SBC YMAX
C4DD  8D 5D 03   STORE3 STA DLTAY3
C4E0  60                RTS
C4E1                    ;
C4E1                    ;****************************************
C4E1                    ;
C4E1                    ; DRAW A SHADED TRIANGULAR FACET
C4E1                    ;
C4E1  20 AE C2   FACET  JSR SORTX
C4E4  AD 47 03          LDA NOSCAL
C4E7  F0 03             BEQ YSOK
C4E9  20 6F C2          JSR SCALE
C4EC  20 6F C4   YSOK   JSR FINDXY
C4EF  AD 4A 03          LDA XMIN
C4F2  8D 3F 03          STA XPLT
C4F5  AD 4B 03          LDA XMIN+1
C4F8  8D 40 03          STA XPLT+1
C4FB  38         FCETLP SEC
C4FC  AD 3F 03          LDA XPLT
C4FF  ED 4A 03          SBC XMIN
C502  8D 5F 03          STA XDIFF
C505  AD 56 03          LDA DLTAX1
C508  F0 53             BEQ CONT
C50A  8D 5A 03          STA DELTAX
C50D  AD 5B 03          LDA DLTAY1
C510  8D 5E 03          STA DELTAY
C513  AD 60 03          LDA FLAG1
C516  8D 63 03          STA FLAG
C519  AD 4C 03          LDA YMIN
C51C  8D 55 03          STA YBASE
C51F  20 3C C4          JSR ENDPTS
C522  8D 53 03          STA YTOP
C525  AD 59 03          LDA DLTAX3
C528  F0 33             BEQ CONT
C52A  8D 5A 03          STA DELTAX
C52D  AD 5D 03          LDA DLTAY3
C530  8D 5E 03          STA DELTAY
C533  AD 62 03          LDA FLAG3
C536  8D 63 03          STA FLAG
C539  20 3C C4          JSR ENDPTS
C53C  8D 54 03          STA YBOT
C53F  20 0D C4          JSR VLINE
C542  AD 40 03          LDA XPLT+1
C545  CD 4E 03          CMP XMID+1
C548  D0 08             BNE NEXTX1
C54A  AD 3F 03          LDA XPLT
C54D  CD 4D 03          CMP XMID
C550  F0 0B             BEQ CONT
C552  EE 3F 03   NEXTX1 INC XPLT
C555  D0 03             BNE SKIP3
C557  EE 40 03          INC XPLT+1
C55A  4C FB C4   SKIP3  JMP FCETLP
C55D  38         CONT   SEC
C55E  AD 3F 03          LDA XPLT
C561  ED 4A 03          SBC XMIN
C564  8D 5F 03          STA XDIFF
C567  AD 59 03          LDA DLTAX3
C56A  F0 63             BEQ FINI
C56C  8D 5A 03          STA DELTAX
C56F  AD 5D 03          LDA DLTAY3
C572  8D 5E 03          STA DELTAY
C575  AD 62 03          LDA FLAG3
C578  8D 63 03          STA FLAG
C57B  AD 4C 03          LDA YMIN
C57E  8D 55 03          STA YBASE
C581  20 3C C4          JSR ENDPTS
C584  8D 54 03          STA YBOT
C587  38                SEC
C588  AD 3F 03          LDA XPLT
C58B  ED 4D 03          SBC XMID
C58E  8D 5F 03          STA XDIFF
C591  AD 58 03          LDA DLTAX2
C594  F0 39             BEQ FINI
C596  8D 5A 03          STA DELTAX
C599  AD 5C 03          LDA DLTAY2
C59C  8D 5E 03          STA DELTAY
C59F  AD 61 03          LDA FLAG2
C5A2  8D 63 03          STA FLAG
C5A5  AD 4F 03          LDA YMID
C5A8  8D 55 03          STA YBASE
C5AB  20 3C C4          JSR ENDPTS
C5AE  8D 53 03          STA YTOP
C5B1  20 0D C4          JSR VLINE
C5B4  AD 40 03          LDA XPLT+1
C5B7  CD 51 03          CMP XMAX+1
C5BA  D0 08             BNE NEXTX2
C5BC  AD 3F 03          LDA XPLT
C5BF  CD 50 03          CMP XMAX
C5C2  F0 0B             BEQ FINI
C5C4  EE 3F 03   NEXTX2 INC XPLT
C5C7  D0 03             BNE SKIP4
C5C9  EE 40 03          INC XPLT+1
C5CC  4C 5D C5   SKIP4  JMP CONT
C5CF  AD 64 03   FINI   LDA EDGES
C5D2  F0 15             BEQ FINISH
C5D4  20 EB C2          JSR OUTLN
C5D7  20 9A C2          JSR SWAP23
C5DA  20 EB C2          JSR OUTLN
C5DD  20 86 C2          JSR SWAP12
C5E0  20 9A C2          JSR SWAP23
C5E3  20 86 C2          JSR SWAP12
C5E6  20 EB C2          JSR OUTLN
C5E9  60         FINISH RTS
C5EA                    .END

ERRORS = 00000

SYMBOL TABLE

SYMBOL VALUE
 CONT     C55D    COUNT    0368    DELTAX   035A    DELTAY   035E
 DIVIDE   C025    DLTAX1   0356    DLTAX2   0358    DLTAX3   0359
 DLTAY1   035B    DLTAY2   035C    DLTAY3   035D    DONE     C43B
 DRAW     C424    DVDND    00FD    DVSOR    00FB    EDGES    0364
 ENDPTS   C43C    ERASE1   C33A    ERASE2   C3B7    ERROR    0365
 FACET    C4E1    FCETLP   C4FB    FINDXY   C46F    FINI     C5CF
 FINISH   C5E9    FLAG     0363    FLAG1    0360    FLAG2    0361
 FLAG3    0362    LINE     C2DB    LNLP1    C32F    LNLP2    C3AC
 LOOP1    C288    LOOP2    C29C    MLPCND   00AC    MLPLER   00AD
 MODE     0367    MULT     C011    NEGSLP   C468    NEXTX1   C552
 NEXTX2   C5C4    NGSLP    C3D1    NOINC1   C357    NOINC2   C3C2
 NORM     C224    NOSCAL   0347    NOSWP1   C2C2    NSLOPE   C347
 ORIGIN   C26F    OUTLN    C2EB    PLOT     C143    PROD     00AE
 QUOT     00FD    RAM      034A    SCALE    C26F    SCLP     C275
 SK1      C33D    SK2      C34A    SK3      C369    SKIP2    C46E
 SKIP3    C55A    SKIP4    C5CC    SKP1     C3BA    SKP2     C3D4
 SKP3     C3E7    SORTED   C2DA    SORTLP   C2B0    SORTX    C2AE
 STEPX    C382    STEPY    C30E    STORE1   C4B3    STORE2   C4C8
 STORE3   C4DD    SWAP12   C286    SWAP23   C29A    TEST     C407

 SYMBOL TABLE

 SYMBOL VALUE
  UNPLOT   C146    VLINE    C40D    XDIFF    035F    XMAX     0350
  XMID     034D    XMIN     034A    XPLT     033F    YBASE    0355
  YBOT     0354    YMAX     0352    YMID     034F    YMIN     034C
  YPLT     0341    YSOK     C4EC    YTOP     0353

 END OF ASSEMBLY
```

End Listing Three

Listing Four

```
0000             ; PRIMITIVE SOLID SHAPE DRAWING
0000             ;
0000             ; RICHARD L. RYLANDER   11/7/84
0000             ;
0000             ; LOAD ARITHMETIC AND GRAPHIC UTILITIES FIRST
0000             ;
0000             ;****************************************
0000             RAM=$036A
0000             ORIGIN=$C5EA
0000             ;
0000             MLPCND=$AC               ; MULTIPLICAND (S)
0000             MLPLER=$AD               ; MULTIPLIER (S)
0000             PROD=$AE                 ; PRODUCT (D)
0000             MULT=$C011               ; CALL FOR MULTIPLY
0000             ;
0000             DVDND=$FD                ; DIVIDEND (D)
```


(Continued on page 70)

ENVOY™

Communications Software

- Easy to use, menu driven, compact and high speed
- Access electronic mail, remote systems and data networks like CompuServe, The Source and Dow Jones News Retrieval
- Terminal mode with large data-capture buffer
- Error-free transfers of text and binary files
- XMODEM (Christensen) and ANSI X3.28 transfer protocols
- User-definable Autodial and Autologin menu
- Utilities menu for file copy, type, print, erase and rename
- Remote unattended file transfers and utilities
- Available for most popular computer systems: IBM PC, PCjr, PC compatibles, Sanyo MBC-55X, MSDOS, CP/M-86, Concurrent CP/M, CP/M 2.2, CP/M Plus and more
- Money-back guarantee if not completely satisfied

$49.95

ARTISOFT INC

P.O. Box 41436
Tucson, AZ 85717
(602) 327-4305

Circle **no. 7** on reader service card.

ConIX™

NOW ONLY $79.95!

If you think you're missing out on innovative software developments because nobody is writing for CP/M™-80, take a look at us. We've adapted UNIX™ features to CP/M like never before, and with the kind of professional, quality-controlled product that you deserve. That product is none other than the critically acclaimed ConIX Operating System.

ConIX can provide any 48K+ CP/M-80 or compatible system with I/O Redirection and Pipes (uses memory or disk), perfected User Areas, Command and Overlay Path Searching, Auto Screen Paging, 8Mb Print Buffering, 22 new SysCalls, Function Keys, "Virtual" disk system, Archiver (saves over 50% disk), extensive command language, 300+ variables, 100+ commands, pull-down menu, and much more! Uses as little as 1/2K RAM! Runs with CP/M for true data and software compatibility. Installs easily without any system mods!

The ConIX package lists at $165 and has been advertised and sold internationally to many enthusiastic customers since October 1983. As a special limited offer, we've lowered the price of the complete ConIX system by 50% to only $79.95! Don't miss this opportunity to bring your 8-bit micro back into the software revolution. Order your copy of ConIX today!

Price includes manual, 8" disk, and user support. 5¼" conversions available. Contact your local dealer, or buy direct and add shipping: $4.50 UPS, $10 Canada, $25 overseas. NY residents add sales tax.

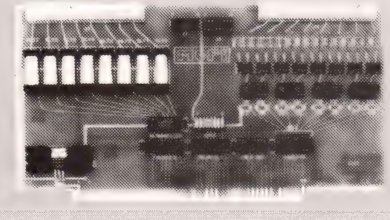
Computer Helper Industries Inc.
P.O. Box 680 Parkchester Station, NY 10462
Tel. (212) 652-1786 (for information/orders)

"We're helping your computer work better for you!"

UNIX: AT&T Bell Labs, CP/M: Digital Research, ConIX: Computer Helper Ind.

Circle **no. 22** on reader service card.

MULLEN S-100: Real Time, Real World CONTROLLER

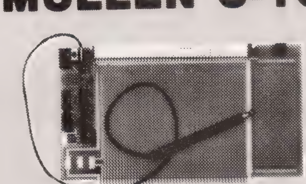


ICB-10 CONTROLLER BOARD
$219, assembled and tested.

This 8 channel digital I/O controller can monitor inputs and control outputs. It features an easy to read manual that has schematics, component list, and programming examples as well as provocative insights on potential applications.

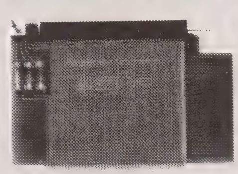
Examples of applications are included in a reprinted article that demonstrates two MULLEN CONTROLLER BOARDS in an interactive system that: feeds a cat; irrigates a garden dependent on soil moisture; closes the window when it rains; controls the thermostat for optimum comfort; controls appliances, lights, security system, and weather monitoring station (logging temperature, wind speed and direction, and graphing pollution content of the atmosphere.) Solenoids, microswitches, pneumatic actuators, pH sensors, and other devices are used in this system.

MULLEN S-100: DEBUGGERS



TB-4a EXTENDER BOARD
The latest in our TB line, the most widely used add-ons in the industry. Features logic probe, formed-lead edge connectors, pulse catcher switch and reset button.

$89, assembled and tested.



ZB-1 ZIF EXTENDER BOARD
This debugger features Zero Insertion Force edge connectors for easy board changes and long life. Expect 2,000 or more insertions rather than the usual 300 to 400 with tension type connectors.
$159, assembled and tested.

MCPI

MULLEN COMPUTER PRODUCTS, INC.

AVAILABLE:

Priority One Electronics,
Chatworth, CA • (213) 709-5111

Jade Computer Products
Hawthorne, CA • (213) 973-7707

E²I Computer Products
Hayward, CA • (415) 786-9203

Mullen Computer Products is the industrial distributer for **CompuPro®**'s products. For more information, call us at (415) 783-2866 or write MCPI, 2260 American Ave., #1, Hayward, CA 94545. OEM sales available from factory.
Prices are subject to change without notice.

Circle **no. 54** on reader service card.

Listing Four

```
0000            DVSOR=$FB              ; DIVISOR (D)
0000            QUOT=$FD               ; QUOTIENT (D)
0000            DIVIDE=$C025           ; CALL FOR DIVIDE
0000            ;
0000            ARG=$AC                ; ARGUMENT (S)
0000            SQR=$AE                ; SQUARE OF ARG (D)
0000            SQUARE=$C004           ; CALL FOR SQUARE
0000            ;
0000            RADCND=$AC             ; RADICAND (D)
0000            ROOT=$033C             ; SQUARE ROOT (S)
0000            SQRT=$C064             ; CALL FOR SQRT
0000            ;
0000            RNDM=$C000             ; RANDOM NUMBER
0000            RANDOM=$C0C8           ; CALL FOR RANDOM
0000            ; NOTE - A CALL TO 'RANDOM' LEAVES A RANDOM BYTE
0000            ; IN THE ACCUMULATOR
0000            ;
0000            XPLT=$033F
0000            YPLT=$0341
0000            NORM=$C224
0000            PLTSHD=$C20F
0000            VALUE=$0344            ; FINAL NORMALIZED SHADE VALUE
0000            HTORRN=$0346           ; SHADE FLAG, 1=HALFTONE
0000            NOSCAL=$0347           ; SCALE FLAG, 1=NO SCALE
0000            ;
0000                   *=RAM
036A            XCENT  *=*+2           ; CENTER COORD
036C            XREL   *=*+1           ; RELATIVE (TO CENTER)
036D            XSHD   *=*+2           ; USED IN SHADE CALC
036F            YCENT  *=*+1           ; CENTER COORD
0370            YREL   *=*+1           ; RELATIVE (TO CENTER)
0371            YSHD   *=*+2           ; USED IN SHADE CALC
0373            ZREL   *=*+2           ; RELATIVE (TO CENTER)
0375            ZWX    *=*+2           ; Z WITH X (+ OR -)
0377            ;
0377            RADIUS *=*+2           ; LOCAL RADIUS OF SURFACE
0379            TONE   *=*+2           ; USED IN SHADE CALC
037B            TNTMP  *=*+2           ; USED IN SHADE CALC
037D            ;
037D            CLIPL  *=*+1           ; LEFT CLIPPING BOUND
037E            CLIPR  *=*+1           ; RIGHT CLIPPING BOUND
037F            CLIPU  *=*+1           ; UP CLIPPING BOUND
0380            CLIPD  *=*+1           ; DOWN CLIPPING BOUND
0381            ;
0381            HEMI   *=*+1           ; PLOTTING HEMISPHERE
0382            ;
0382            BAKLIT *=*+1           ; BACKLIT FLAG
0383            HVFLAG *=*+1           ; HORIZONTAL/VERTICAL FLAG
0384            TEMP   *=*+2           ; TEMPORARY STORAGE
0386            CNTX   *=*+1           ; LOOP COUNTER
0387            CNTY   *=*+1           ; LOOP COUNTER
0388            MAX    *=*+1           ; LOOP LIMIT
0389            ;
0389            HLEN   *=*+1           ; HALF LENGTH OF CYLINDERS
038A            RS     *=*+2           ; SQUARE OF TOROID RADIUS
038C            RT     *=*+1           ; TOROID (RING) RADIUS
038D            RC     *=*+1           ; CENTER RADIUS OF TOROID
038E            RO     *=*+1           ; OUTER RADIUS OF TOROID
038F            RI     *=*+1           ; INNER RADIUS OF TOROID
0390            XSQR   *=*+2
0392            XMAX   *=*+1
0393            ;
0393            R0=HLEN
0393            ;
0393                   *=ORIGIN
C5EA            ;****************************************
C5EA            ;
C5EA            ; DIVIDE WITH SINGLE PRECISION DIVISOR
C5EA            ; (USED OFTEN IN SHAPE ROUTINES)
C5EA            ;
C5EA  A9 00     SDIV   LDA #0
C5EC  85 FC            STA DVSOR+1
C5EE  4C 25 C0         JMP DIVIDE
C5F1            ;
C5F1            ;****************************************
C5F1            ;
C5F1            ; CALCULATE SHADE VALUE (0-63) BY
C5F1            ; MULTIPLYING 'TONE' BY 26 THEN
C5F1            ; DIVIDE RESULT BY RADIUS OF SURFACE
C5F1            ;
C5F1  2C 7A 03  GETVAL BIT TONE+1
C5F4  10 12            BPL CNTNU       ; IF 'TONE'<0, THEN
C5F6  AD 82 03         LDA BAKLIT      ; MAKE VALUE 0 OR ABS(TONE)
C5F9  D0 04            BNE NEGATE      ; DEPENDING ON BAKLIT FLAG
C5FB  8D 44 03         STA VALUE
C5FE  60               RTS
C5FF  38        NEGATE SEC
C600  A9 00            LDA #$00
C602  ED 79 03         SBC TONE
C605  8D 79 03         STA TONE
C608  AD 79 03  CNTNU  LDA TONE
C60B  85 AC            STA MLPCND
C60D  A9 1A            LDA #$1A
C60F  85 AD            STA MLPLER
C611  20 11 C0         JSR MULT
C614  85 FE            STA DVDND+1
C616  A5 AE            LDA PROD
C618  85 FD            STA DVDND
C61A  AD 77 03         LDA RADIUS
C61D  85 FB            STA DVSOR
C61F  20 EA C5         JSR SDIV
C622  A5 FD            LDA QUOT
C624  8D 44 03         STA VALUE
C627  60               RTS
C628            ;
C628            ;****************************************
C628            ;
C628            ; POINT PLOTTING BY QUADRANTS USING
C628            ; THE FOUR-FOLD SYMMETRY OF SIMPLE OBJECTS
C628            ;
C628            ; DEPENDING ON STATUS OF 'HVFLAG', EXCHANGE
C628            ; X AND Y COORDINATES TO ROTATE OBJECTS 90 DEG
C628            ; SINGLE SHAPE ROUTINE CAN THEN BE USED TO
C628            ; DRAW 'HORIZONTAL' OR 'VERTICAL' VERSIONS
C628            ; OF AN OBJECT
C628            ;
C628            ; THE FOLOWING IS A 'BASIC SUBROUTINE'
C628            ; EQUIVALENT TO EXPLAIN ITS OPERATION
C628            ;
C628            ; NOTE THAT LABELS ARE USED IN PLACE OF
C628            ; LINE NUMBERS
C628            ;
C628            ; 'PTPLOT' IF HVFLAG<0 THEN GOTO 'NOROT'
C628            ;          (STACK)=XREL:XREL=YREL:YREL=(STACK)
C628            ;          (STACK)=XSHD:XSHD=YSHD:YSHD=(STACK)
C628            ; 'NOROT'  GOSUB 'GETZ'
C628            ;          REM CALCULATE 2*Z FROM X,Y AND RADIUS
C628            ;          HEMI = 1
C628            ;          IF XREL>CLIPL THEN GOTO 'RHEMI'
C628            ;          ZWX=2*Z-XSHD
C628            ;          XPLT=XCENT-XREL:REM LEFT HEMISPHERE
C628            ; 'CHCLUP' IF YREL>CLIPU THEN GOTO 'DHEMI'
C628            ;          TONE=ZWX+YSHD
C628            ;          GOSUB 'GETVAL':REM NORMALIZE SHADE VAL
C628            ;          YPLT=YCENT+YREL
C628            ;          GOSUB 'PLTSHD':REM PLOT OR UNPLOT
C628            ;          REM   POINTS WEIGHTED BY SHADE VALUE
C628            ; 'DHEMI'  IF YREL>CLIPD THEN GOTO 'RHEMI'
C628            ;          TONE=ZWX-YSHD
C628            ;          GOSUB 'GETVAL'
C628            ;          YPLT=YCENT-YREL
C628            ;          GOSUB 'PLTSHD'
C628            ; 'RHEMI'  IF HEMI=0 THEN RETURN
C628            ;          HEMI=0
C628            ;          IF XREL>CLIPR THEN RETURN
C628            ;          ZWX=2*Z+XSHD
C628            ;          XPLT=XCENT+XREL
C628            ;          GOSUB 'CHCLUP'
C628            ;          RETURN
C628            ;
C628  2C 83 03  PTPLOT BIT HVFLAG
C62B  10 2D            BPL NOROT
C62D  AD 6C 03         LDA XREL
C630  48               PHA
C631  48               PHA
C632  AD 70 03         LDA YREL
C635  8D 6C 03         STA XREL
C638  68               PLA
C639  8D 70 03         STA YREL
C63C  AD 6D 03         LDA XSHD
C63F  48               PHA
C640  48               PHA
C641  AD 71 03         LDA YSHD
C644  8D 6D 03         STA XSHD
C647  68               PLA
C648  8D 71 03         STA YSHD
C64B  AD 6E 03         LDA XSHD+1
C64E  48               PHA
C64F  48               PHA
C650  AD 72 03         LDA YSHD+1
C653  8D 6E 03         STA XSHD+1
C656  68               PLA
C657  8D 72 03         STA YSHD+1
C65A  20 45 C7  NOROT  JSR GETZ
C65D  A9 01     PTPLT2 LDA #$01
C65F  8D 81 03         STA HEMI
C662  38               SEC
C663  AD 7D 03         LDA CLIPL       ; CHECK LEFT HEMISPHERE
C666  CD 6C 03         CMP XREL
C669  90 7D            BCC RHEMI
C66B  38               SEC
C66C  AD 3C 03         LDA ROOT
C66F  ED 6D 03         SBC XSHD
C672  8D 75 03         STA ZWX
C675  AD 3D 03         LDA ROOT+1
C678  ED 6E 03         SBC XSHD+1
C67B  8D 76 03         STA ZWX+1
C67E  38               SEC
C67F  AD 6A 03         LDA XCENT
C682  ED 6C 03         SBC XREL
C685  8D 3F 03         STA XPLT
C688  AD 6B 03         LDA XCENT+1
C68B  E9 00            SBC #$00
C68D  8D 40 03         STA XPLT+1
C690            ;
C690  38        CHCLUP SEC
C691  AD 7F 03         LDA CLIPU       ; CHECK FOR UP CLIPPING
C694  CD 70 03         CMP YREL
C697  90 23            BCC DHEMI
C699  18               CLC
C69A  AD 75 03         LDA ZWX
C69D  6D 71 03         ADC YSHD
C6A0  8D 79 03         STA TONE
C6A3  AD 76 03         LDA ZWX+1
C6A6  6D 72 03         ADC YSHD+1
C6A9  8D 7A 03         STA TONE+1
C6AC  20 F1 C5         JSR GETVAL
C6AF  18               CLC
C6B0  AD 6F 03         LDA YCENT
C6B3  6D 70 03         ADC YREL
C6B6  8D 41 03         STA YPLT
C6B9  20 0F C2         JSR PLTSHD
C6BC            ;
C6BC  38        DHEMI  SEC
C6BD  AD 80 03         LDA CLIPD       ; CHECK FOR DOWN CLIPPING
C6C0  CD 70 03         CMP YREL
C6C3  90 23            BCC RHEMI
C6C5  38               SEC
C6C6  AD 75 03         LDA ZWX
C6C9  ED 71 03         SBC YSHD
C6CC  8D 79 03         STA TONE
C6CF  AD 76 03         LDA ZWX+1
C6D2  ED 72 03         SBC YSHD+1
C6D5  8D 7A 03         STA TONE+1
C6D8  20 F1 C5         JSR GETVAL
C6DB  38               SEC
C6DC  AD 6F 03         LDA YCENT
C6DF  ED 70 03         SBC YREL
C6E2  8D 41 03         STA YPLT
C6E5  20 0F C2         JSR PLTSHD
C6E8            ;
C6E8  AD 81 03  RHEMI  LDA HEMI
C6EB  F0 34            BEQ PLDONE
C6ED  CE 81 03         DEC HEMI
C6F0  38               SEC
C6F1  AD 7E 03         LDA CLIPR       ; CHECK FOR RIGHT CLIPPING
C6F4  CD 6C 03         CMP XREL
C6F7  90 28            BCC PLDONE
C6F9  18               CLC
C6FA  AD 3C 03         LDA ROOT
C6FD  6D 6D 03         ADC XSHD
C700  8D 75 03         STA ZWX
C703  AD 3D 03         LDA ROOT+1
C706  6D 6E 03         ADC XSHD+1
C709  8D 76 03         STA ZWX+1
C70C  18               CLC
C70D  AD 6A 03         LDA XCENT
C710  6D 6C 03         ADC XREL
C713  8D 3F 03         STA XPLT
C716  AD 6B 03         LDA XCENT+1
C719  69 00            ADC #$00
C71B  8D 40 03         STA XPLT+1
C71E  4C 90 C6         JMP CHCLUP
C721  2C 83 03  PLDONE BIT HVFLAG
C724  10 1E            BPL NORSTR
```

(Continued on page 72)

Listing Four

```
C726  AD 6E 03        LDA XSHD+1      ; RESTORE COORDS
C729  8D 72 03        STA YSHD+1
C72C  68              PLA
C72D  8D 6E 03        STA XSHD+1
C730  AD 6D 03        LDA XSHD
C733  8D 71 03        STA YSHD
C736  68              PLA
C737  8D 6D 03        STA XSHD
C73A  AD 6C 03        LDA XREL
C73D  8D 70 03        STA YREL
C740  68              PLA
C741  8D 6C 03        STA XREL
C744  60       NORSTR RTS
C745           ;
C745           ;****************************************
C745           ;
C745           ; CALCULATE Z FROM LOCAL X,Y BY
C745           ; PYTHAGOREAN SUM
C745           ;
C745  AD 77 03 GETZ   LDA RADIUS
C748  85 AC           STA ARG
C74A  20 04 C0        JSR SQUARE
C74D  8D 7C 03        STA TNTMP+1
C750  A5 AE           LDA SQR
C752  8D 7B 03        STA TNTMP
C755  AD 6D 03        LDA XSHD
C758  85 AC           STA ARG
C75A  20 04 C0        JSR SQUARE
C75D  38              SEC
C75E  AD 7B 03        LDA TNTMP
C761  E5 AE           SBC SQR
C763  8D 7B 03        STA TNTMP
C766  AD 7C 03        LDA TNTMP+1
C769  E5 AF           SBC SQR+1
C76B  8D 7C 03        STA TNTMP+1
C76E  AD 71 03        LDA YSHD
C771  85 AC           STA ARG
C773  20 04 C0        JSR SQUARE
C776  38              SEC
C777  AD 7B 03        LDA TNTMP
C77A  E5 AE           SBC SQR
C77C  85 AC           STA RADCND
C77E  AD 7C 03        LDA TNTMP+1
C781  E5 AF           SBC SQR+1
C783  85 AD           STA RADCND+1
C785  30 0A           BMI ZEROOT
C787  20 64 C0        JSR SQRT
C78A  0E 3C 03        ASL ROOT
C78D  2E 3D 03        ROL ROOT+1
C790  60              RTS
C791  A9 00    ZEROOT LDA #$00
C793  8D 3C 03        STA ROOT
C796  8D 3D 03        STA ROOT+1
C799  60              RTS
C79A           ;
C79A           ;****************************************
C79A           ;
C79A           ; SET UP PARAMETERS FOR TOROIDS
C79A           ;
C79A           ; RT=(RO-RI)/2   RS=RT*RT   RC=RT+RI
C79A           ;
C79A  AD 8E 03 TPARM  LDA RO
C79D  38              SEC
C79E  ED 8F 03        SBC RI
C7A1  4A              LSR A
C7A2  8D 8C 03        STA RT
C7A5  8D 77 03        STA RADIUS
C7A8  18              CLC
C7A9  6D 8F 03        ADC RI
C7AC  8D 8D 03        STA RC
C7AF  AD 8C 03        LDA RT
C7B2  85 AC           STA ARG
C7B4  20 04 C0        JSR SQUARE
C7B7  A5 AE           LDA SQR
C7B9  8D 8A 03        STA RS
C7BC  A5 AF           LDA SQR+1
C7BE  8D 8B 03        STA RS+1
C7C1  A9 00           LDA #0
C7C3  8D 86 03        STA CNTX
C7C6  60              RTS
C7C7           ;
C7C7           ;****************************************
C7C7           ;
C7C7           ; DRAW A SHADED SPHERE
C7C7           ;
C7C7           ; 'BASIC SUBROUTINE' EQUIVALENT
C7C7           ;
C7C7           ; 'SPHERE' FOR CNTX=0 TO RADIUS/SQR(2)
C7C7           ;          XREL=CNTX:XSHD=CNTX
C7C7           ;          FOR CNTY=CNTX TO SQR(RAD*RAD-CNTX*CNTX)
C7C7           ;          YREL=CNTY:YSHD=CNTY
C7C7           ;          HVFLAG=0
C7C7           ;          GOSUB 'PTPLOT'
C7C7           ;          REM EXCHANGE X & Y TO USE 8-FOLD SYM
C7C7           ;          HVFLAG=-128
C7C7           ;
C7C7           ;          GOSUB 'PTPLOT'
C7C7           ;          NEXT CNTY
C7C7           ;          NEXT CNTX
C7C7           ;          RETURN
C7C7           ;
C7C7           ;
C7C7  AD 77 03 SPHERE LDA RADIUS
C7CA  85 AC           STA ARG
C7CC  20 04 C0        JSR SQUARE
C7CF  06 AE           ASL SQR
C7D1  26 AF           ROL SQR+1
C7D3  A5 AE           LDA SQR
C7D5  85 AC           STA RADCND
C7D7  A5 AF           LDA SQR+1
C7D9  85 AD           STA RADCND+1
C7DB  20 64 C0        JSR SQRT
C7DE  4E 3D 03        LSR ROOT+1
C7E1  6E 3C 03        ROR ROOT
C7E4  AD 3C 03        LDA ROOT
C7E7  8D 92 03        STA XMAX
C7EA  A9 00           LDA #$00
C7EC  8D 86 03        STA CNTX
C7EF  8D 6E 03        STA XSHD+1
C7F2  8D 72 03        STA YSHD+1
C7F5  AD 77 03        LDA RADIUS
C7F8  85 AC           STA ARG
C7FA  20 04 C0        JSR SQUARE
C7FD  8D 85 03        STA TEMP+1
C800  A5 AE           LDA SQR
C802  8D 84 03        STA TEMP
C805  AD 86 03 LOOPX  LDA CNTX
C808  8D 87 03        STA CNTY
C80B  85 AC           STA ARG
C80D  8D 6C 03        STA XREL
C810  8D 6D 03        STA XSHD
C813  20 04 C0        JSR SQUARE
C816  38              SEC
C817  AD 84 03        LDA TEMP
C81A  E5 AE           SBC SQR
C81C  85 AC           STA RADCND
C81E  AD 85 03        LDA TEMP+1
C821  E5 AF           SBC SQR+1
C823  85 AD           STA RADCND+1
C825  20 64 C0        JSR SQRT
C828  AD 3C 03        LDA ROOT
C82B  8D 88 03        STA MAX
C82E  AD 87 03 LOOPY  LDA CNTY
C831  8D 70 03        STA YREL
C834  8D 71 03        STA YSHD
C837  A9 00           LDA #0
C839  8D 83 03        STA HVFLAG
C83C  20 28 C6        JSR PTPLOT
C83F  A9 80           LDA #$80
C841  8D 83 03        STA HVFLAG
C844  20 28 C6        JSR PTPLOT
C847  AD 87 03        LDA CNTY
C84A  CD 88 03        CMP MAX
C84D  F0 06           BEQ DONEY
C84F  EE 87 03        INC CNTY
C852  4C 2E C8        JMP LOOPY
C855  AD 86 03 DONEY  LDA CNTX
C858  CD 92 03        CMP XMAX
C85B  F0 06           BEQ DONE
C85D  EE 86 03        INC CNTX
C860  4C 05 C8        JMP LOOPX
C863  60       DONE   RTS
C864           ;
C864           ;****************************************
C864           ;
C864           ; DRAW SHADED CYLINDERS
C864           ;
C864           ; 'BASIC SUBROUTINE' EQUIVALENT
C864           ;
C864           ; 'CYLNDR' XSHD=0
C864           ;          FOR YREL=RADIUS TO 0
C864           ;          YSHD=YREL
C864           ;          FOR XREL=HLEN TO 0
C864           ;          GOSUB 'PTPLOT'
C864           ;          NEXT XREL
C864           ;          NEXT YREL
C864           ;          RETURN
C864           ;
C864  A9 00    CYLNDR LDA #0
C866  8D 6D 03        STA XSHD
C869  8D 6E 03        STA XSHD+1
C86C  8D 72 03        STA YSHD+1
C86F  AD 77 03        LDA RADIUS
C872  8D 70 03        STA YREL
C875  AD 89 03 CYLOOP LDA HLEN
C878  8D 6C 03        STA XREL
C87B  AD 70 03        LDA YREL
C87E  8D 71 03        STA YSHD
C881  20 28 C6 CXLOOP JSR PTPLOT
C884  CE 6C 03        DEC XREL
C887  10 F8           BPL CXLOOP
C889  CE 70 03        DEC YREL
C88C  10 E7           BPL CYLOOP
C88E  60              RTS
C88F           ;
C88F           ;****************************************
C88F           ;
C88F           ; DRAW EDGE-VIEW TOROIDS
C88F           ;
C88F           ; 'BASIC SUBROUTINE' EQUIVALENT
C88F           ;
C88F           ; 'EDGTOR' GOSUB 'TPARM':REM SET UP RADII
C88F           ;          FOR CNTX=0 TO RT
C88F           ;          XREL=CNTX:XSHD=CNTX
C88F           ;          R0=SQR(RT*RT-CNTX*CNTX)
C88F           ;          FOR CNTY=0 TO R0+RC
C88F           ;          YREL=CNTY
C88F           ;          YSHD=(R0*CNTY)/(R0+RC)
C88F           ;          GOSUB 'PTPLOT'
C88F           ;          NEXT CNTY
C88F           ;          NEXT CNTX
C88F           ;          RETURN
C88F           ;
C88F  20 9A C7 EDGTOR JSR TPARM
C892  A9 00           LDA #$00
C894  8D 6E 03        STA XSHD+1
C897  8D 72 03        STA YSHD+1
C89A  AD 86 03 LOOPX4 LDA CNTX
C89D  8D 6C 03        STA XREL
C8A0  8D 6D 03        STA XSHD
C8A3  85 AC           STA ARG
C8A5  20 04 C0        JSR SQUARE
C8A8  38              SEC
C8A9  AD 8A 03        LDA RS
C8AC  E5 AE           SBC SQR
C8AE  85 AC           STA RADCND
C8B0  AD 8B 03        LDA RS+1
C8B3  E5 AF           SBC SQR+1
C8B5  85 AD           STA RADCND+1
C8B7  20 64 C0        JSR SQRT
C8BA  AD 3C 03        LDA ROOT
C8BD  8D 89 03        STA R0
C8C0  18              CLC
C8C1  6D 8D 03        ADC RC
C8C4  8D 88 03        STA MAX
C8C7  A9 00           LDA #$00
C8C9  8D 87 03        STA CNTY
C8CC  AD 87 03 LOOPY4 LDA CNTY
C8CF  8D 70 03        STA YREL
C8D2  85 AD           STA MLPLER
C8D4  AD 89 03        LDA R0
C8D7  85 AC           STA MLPCND
C8D9  20 11 C0        JSR MULT
C8DC  85 FE           STA DVDND+1
C8DE  A5 AE           LDA PROD
C8E0  85 FD           STA DVDND
C8E2  AD 88 03        LDA MAX
C8E5  85 FB           STA DVSOR
C8E7  20 EA C5        JSR SDIV
C8EA  A5 FD           LDA QUOT
C8EC  8D 71 03        STA YSHD
C8EF  20 28 C6        JSR PTPLOT
C8F2  AD 87 03        LDA CNTY
C8F5  CD 88 03        CMP MAX
C8F8  F0 06           BEQ DONE4
C8FA  EE 87 03        INC CNTY
```

(Continued on page 74)

Listing Four

```
C8FD  4C CC C8          JMP LOOPY4
C900  AD 86 03  DONE4   LDA CNTX
C903  CD 8C 03          CMP RT
C906  F0 06             BEQ DONEHT
C908  EE 86 03          INC CNTX
C90B  4C 9A C8          JMP LOOPX4
C90E  60        DONEHT  RTS
C90F                    ;
C90F                    ;*****************************************
C90F                    ;
C90F                    ; DRAW A SHADED, TOP-VIEW TOROID
C90F                    ;
C90F                    ; 'BASIC SUBROUTINE' EQUIVALENT
C90F                    ;
C90F                    ; 'TOROID' GOSUB 'TPARM'
C90F                    ;          FOR CNTX=0 TO RO/SQR(2)
C90F                    ;          REM 8-FOLD SYMMETRY USED
C90F                    ;          XREL=CNTX
C90F                    ;          MAX=SQR(RO*RO-CNTX*CNTX)
C90F                    ;          IF CNTX>RI THEN GOTO 'GRTR'
C90F                    ;          CNTY=SQR(RI*RI-CNTY*CNTY)
C90F                    ;          GOTO 'LLPY1'
C90F                    ; 'GRTR'   CNTY=CNTX
C90F                    ; 'LLPY1'  YREL=CNTY
C90F                    ;          R0=SQR(CNTY*CNTY+CNTX*CNTX)
C90F                    ;          XSHD=CNTX-(CNTX*RC)/R0
C90F                    ;          YSHD=CNTY-(CNTY*RC)/R0
C90F                    ;          HVFLAG=0:GOSUB 'PTPLOT'
C90F                    ;          HVFLAG=-128:GOSUB 'PTPLOT'
C90F                    ;          IF CNTY=MAX THEN GOTO 'DDNY1'
C90F                    ;          CNTY=CNTY+1
C90F                    ;          GOTO 'LLPY1'
C90F                    ; 'DDNY1'  NEXT CNTX
C90F                    ;          RETURN
C90F                    ;
C90F  20 9A C7  TOROID  JSR TPARM
C912  AD 8E 03          LDA RO
C915  85 AC             STA ARG
C917  20 04 C0          JSR SQUARE
C91A  06 AE             ASL SQR
C91C  26 AF             ROL SQR+1
C91E  A5 AE             LDA SQR
C920  85 AC             STA RADCND
C922  A5 AF             LDA SQR+1
C924  85 AD             STA RADCND+1
C926  20 64 C0          JSR SQRT
C929  4E 3D 03          LSR ROOT+1
C92C  6E 3C 03          ROR ROOT
C92F  AD 3C 03          LDA ROOT
C932  8D 92 03          STA XMAX
C935  AD 86 03  LLPX1   LDA CNTX
C938  8D 6C 03          STA XREL
C93B  85 AC             STA ARG
C93D  20 04 C0          JSR SQUARE
C940  8D 91 03          STA XSQR+1
C943  A5 AE             LDA SQR
C945  8D 90 03          STA XSQR
C948  AD 8E 03          LDA RO
C94B  85 AC             STA ARG
C94D  20 04 C0          JSR SQUARE
C950  38                SEC
C951  A5 AE             LDA SQR
C953  ED 90 03          SBC XSQR
C956  85 AC             STA RADCND
C958  A5 AF             LDA SQR+1
C95A  ED 91 03          SBC XSQR+1
C95D  85 AD             STA RADCND+1
C95F  20 64 C0          JSR SQRT
C962  AD 3C 03          LDA ROOT
C965  8D 88 03          STA MAX
C968  38                SEC
C969  AD 8F 03          LDA RI
C96C  ED 86 03          SBC CNTX
C96F  90 23             BCC GRTR
C971  AD 8F 03          LDA RI
C974  85 AC             STA ARG
C976  20 04 C0          JSR SQUARE
C979  38                SEC
C97A  A5 AE             LDA SQR
C97C  ED 90 03          SBC XSQR
C97F  85 AC             STA RADCND
C981  A5 AF             LDA SQR+1
C983  ED 91 03          SBC XSQR+1
C986  85 AD             STA RADCND+1
C988  20 64 C0          JSR SQRT
C98B  AD 3C 03          LDA ROOT
C98E  8D 87 03          STA CNTY
C991  4C 9A C9          JMP LLPY1
C994  AD 86 03  GRTR    LDA CNTX
C997  8D 87 03          STA CNTY
C99A  AD 87 03  LLPY1   LDA CNTY
C99D  8D 70 03          STA YREL
C9A0  85 AC             STA ARG
C9A2  20 04 C0          JSR SQUARE
C9A5  18                CLC
C9A6  A5 AE             LDA SQR
C9A8  6D 90 03          ADC XSQR
C9AB  85 AC             STA RADCND
C9AD  A5 AF             LDA SQR+1
C9AF  6D 91 03          ADC XSQR+1
C9B2  85 AD             STA RADCND+1
C9B4  20 64 C0          JSR SQRT
C9B7  AD 3C 03          LDA ROOT
C9BA  8D 89 03          STA R0
C9BD  85 FB             STA DVSOR
C9BF  AD 86 03          LDA CNTX
C9C2  85 AD             STA MLPLER
C9C4  AD 8D 03          LDA RC
C9C7  85 AC             STA MLPCND
C9C9  20 11 C0          JSR MULT
C9CC  85 FE             STA DVDND+1
C9CE  A5 AE             LDA PROD
C9D0  85 FD             STA DVDND
C9D2  20 EA C5          JSR SDIV
C9D5  38                SEC
C9D6  AD 86 03          LDA CNTX
C9D9  E5 FD             SBC QUOT
C9DB  8D 6D 03          STA XSHD
C9DE  A9 00             LDA #$00
C9E0  E5 FE             SBC QUOT+1
C9E2  8D 6E 03          STA XSHD+1
C9E5  AD 87 03          LDA CNTY
C9E8  85 AD             STA MLPLER
C9EA  AD 8D 03          LDA RC
C9ED  85 AC             STA MLPCND
C9EF  20 11 C0          JSR MULT
C9F2  85 FE             STA DVDND+1
C9F4  A5 AE             LDA PROD
C9F6  85 FD             STA DVDND
C9F8  AD 89 03          LDA R0
C9FB  85 FB             STA DVSOR
C9FD  20 EA C5          JSR SDIV
CA00  38                SEC
CA01  AD 87 03          LDA CNTY
CA04  E5 FD             SBC QUOT
CA06  8D 71 03          STA YSHD
CA09  A9 00             LDA #$00
CA0B  8D 83 03          STA HVFLAG
CA0E  E5 FE             SBC QUOT+1
CA10  8D 72 03          STA YSHD+1
CA13  20 28 C6          JSR PTPLOT
CA16  A9 80             LDA #$80
CA18  8D 83 03          STA HVFLAG
CA1B  20 28 C6          JSR PTPLOT
CA1E  AD 87 03          LDA CNTY
CA21  CD 88 03          CMP MAX
CA24  F0 06             BEQ DDNY1
CA26  EE 87 03          INC CNTY
CA29  4C 9A C9          JMP LLPY1
CA2C  AD 86 03  DDNY1   LDA CNTX
CA2F  CD 92 03          CMP XMAX
CA32  F0 06             BEQ DUNTOR
CA34  EE 86 03          INC CNTX
CA37  4C 35 C9          JMP LLPX1
CA3A  60        DUNTOR  RTS
CA3B                    ;
CA3B                    ;*****************************************
CA3B                    ;
CA3B                    ; DRAW "INSIDE VIEW" TOROIDS
CA3B                    ;
CA3B                    ; 'BASIC SUBROUTINE' EQUIVALENT
CA3B                    ;
CA3B                    ; 'SPOOL'  GOSUB 'TPARM'
CA3B                    ;          FOR CNTX=0 TO RT
CA3B                    ;          XREL=CNTX:XSHD=CNTX
CA3B                    ;          MAX=RC-SQR(RS-CNTX*CNTX)
CA3B                    ;          FOR CNTY=0 TO MAX
CA3B                    ;          YREL=CNTY
CA3B                    ;          YSHD=(RC*CNTY/MAX)-CNTY
CA3B                    ;          GOSUB 'PTPLOT'
CA3B                    ;          NEXT CNTY
CA3B                    ;          NEXT CNTX
CA3B                    ;          RETURN
CA3B                    ;
CA3B  20 9A C7  SPOOL   JSR TPARM
CA3E  AD 86 03  LLPX2   LDA CNTX
CA41  8D 6C 03          STA XREL
CA44  85 AC             STA ARG
CA46  38                SEC
CA47  A9 00             LDA #$00
CA49  ED 86 03          SBC CNTX
CA4C  8D 6D 03          STA XSHD
CA4F  A9 00             LDA #$00
CA51  E9 00             SBC #$00
CA53  8D 6E 03          STA XSHD+1
CA56  20 04 C0          JSR SQUARE
CA59  38                SEC
CA5A  AD 8A 03          LDA RS
CA5D  E5 AE             SBC SQR
CA5F  85 AC             STA RADCND
CA61  AD 8B 03          LDA RS+1
CA64  E5 AF             SBC SQR+1
CA66  85 AD             STA RADCND+1
CA68  20 64 C0          JSR SQRT
CA6B  38                SEC
CA6C  AD 8D 03          LDA RC
CA6F  ED 3C 03          SBC ROOT
CA72  8D 88 03          STA MAX
CA75  A9 00             LDA #$00
CA77  8D 87 03          STA CNTY
CA7A  AD 87 03  LLPY2   LDA CNTY
CA7D  8D 70 03          STA YREL
CA80  85 AD             STA MLPLER
CA82  AD 8D 03          LDA RC
CA85  85 AC             STA MLPCND
CA87  20 11 C0          JSR MULT
CA8A  85 FE             STA DVDND+1
CA8C  A5 AE             LDA PROD
CA8E  85 FD             STA DVDND
CA90  AD 88 03          LDA MAX
CA93  85 FB             STA DVSOR
CA95  20 EA C5          JSR SDIV
CA98  A5 FD             LDA QUOT
CA9A  38                SEC
CA9B  ED 87 03          SBC CNTY
CA9E  8D 71 03          STA YSHD
CAA1  A5 FE             LDA QUOT+1
CAA3  E9 00             SBC #$00
CAA5  8D 72 03          STA YSHD+1
CAA8  20 28 C6          JSR PTPLOT
CAAB  AD 87 03          LDA CNTY
CAAE  CD 88 03          CMP MAX
CAB1  F0 06             BEQ DDNY2
CAB3  EE 87 03          INC CNTY
CAB6  4C 7A CA          JMP LLPY2
CAB9  AD 86 03  DDNY2   LDA CNTX
CABC  CD 8C 03          CMP RT
CABF  F0 06             BEQ DUNHSP
CAC1  EE 86 03          INC CNTX
CAC4  4C 3E CA          JMP LLPX2
CAC7  60        DUNHSP  RTS
CAC8                    .END

ERRORS = 00000

SYMBOL TABLE

SYMBOL VALUE
 ARG      00AC    BAKLIT   0382    CHCLUP   C690    CLIPD    0380
 CLIPL    037D    CLIPR    037E    CLIPU    037F    CNTNU    C608

SYMBOL TABLE

SYMBOL VALUE
 CNTX     0386    CNTY     0387    CXLOOP   C881    CYLNDR   C864
 CYLOOP   C875    DDNY1    CA2C    DDNY2    CAB9    DHEMI    C6BC
 DIVIDE   C025    DONE     C863    DONE4    C900    DONEHT   C90E
 DONEY    C855    DUNHSP   CAC7    DUNTOR   CA3A    DVDND    00FD
 DVSOR    00FB    EDGTOR   C88F    GETVAL   C5F1    GETZ     C745
 GRTR     C994    HEMI     0381    HLEN     0389    HTORRN   0346
 HVFLAG   0383    LLPX1    C935    LLPX2    CA3E    LLPY1    C99A
 LLPY2    CA7A    LOOPX    C805    LOOPX4   C89A    LOOPY    C82E
 LOOPY4   C8CC    MAX      0388    MLPCND   00AC    MLPLER   00AD
 MULT     C011    NEGATE   C5FF    NORM     C224    NOROT    C65A
```

End Listing Four

(Listing Five begins on page 76)

BIG DISCOUNTS FROM JOHN D. OWENS ASSOCIATES

MACROTECH MI-286: 80286/Z-80H DUAL PROCESSOR S-100 CPU BOARD: $1,116
MACROTECH MEMORY MSR: 120NS, high-speed dynamic RAM with realistic pricing:
Works with CompuPro 8085/8088; MT-286 and others:
256K: $556 512K: $876
MACROTECH STATIC RAM: Substitute for RAM 22 and RAM 23.
256K STATIC: $960 512K STATIC: $1,800

EMERALD SYSTEMS HARD DISK SUBSYSTEMS and TAPE BACKUP
High capacity! Up to 280 MB! Emerald has overcome the 32MB DOS limitation! Allows multiple volumes per physical drive. Back up and restore utilities. Ideal for LAN applications.

HOUSTON INSTRUMENTS: Plotters: DMP 41 OR 42: $2,397; DMP 29: $1,838
DIGITIZERS: DT11 $694; DT114 $750; DT11AA $714
New! 14 pen DMP 51 and 52: $4,796

ILLUMINATED TECHNOLOGY S-100 COLOR GRAPHICS: $1,116

NEC APC III: 80186 MS-DOS system w/spectacular graphics: 20% off list

---

SEMIDISK 2MB DISK EMULATOR FOR IBM PC and EPSON QX 10: $2,050

BIG DISCOUNTS on **LOMAS, COMPUPRO, IMS, INTERCONTINENTAL, DIGITAL GRAPHICS SYSTEMS, ADVANCED DIGITAL, ACKERMAN, BYAD** and many others.
Prices & availability subject to change without notice.

Write or call for product literature and inventory sale list.

WE EXPORT: OVERSEAS CALLERS: TWX 710 588 2844 (OWENSASSOC NYK)

DOMESTIC AND OVERSEAS ORDERS TAKEN BY PHONE, LETTER, TELEX OR EASY LINK.
We accept VISA and Mastercard. Shipping $5 per board in continental USA.

JOHN D. OWENS ASSOCIATES
12 SCHUBERT STREET STATEN ISLAND, NEW YORK 10305
(718) 448 6283 (718) 448 6298 (718) 448 2913
EASY LINK MAILBOX ADDRESS: 63840768

LATTICE® WORKS

LATTICE UNVEILS FOUR PRODUCTS

Lattice has announced the availability of four new software products for MS-DOS environments:

C-SPRITE™ is a software tool that simplifies debugging of programs written in Lattice C or assembly language. Cost: $175 per copy.

LMK™ is an Automated Product Generation Utility (UNIX "MAKE") that enhances productivity and relieves the tedium of rebuilding complex software systems or documents. Cost: $195 per copy.

The TEXT MANAGEMENT UTILITY PACKAGE™ includes utilities to search a set of files for simple or complicated patterns, to see the exact minimal differences between two text files, and to modify one or more text files automatically. Cost: $120 per copy.

CVUE™ is a full screen text editor that supports all normal screen editor functions and includes a configuration program to define tabstop positions, horizontal scrolling and edit commands. Cost: $100 per copy.

For complete information on these new products, contact Lattice.

LATTICE C NAMED 'BEST OF 1984'

The Lattice C compiler has been rated 'Best of 1984' by PC Magazine. According to columnist Peter Norton, "The Lattice C compiler is quite good . . . and in my opinion noticeably better than any of its competitors. Lattice C generates code that is quite compact and fast running; the closest competitor in my tests generated code that was about 10 to 15 percent bulkier."

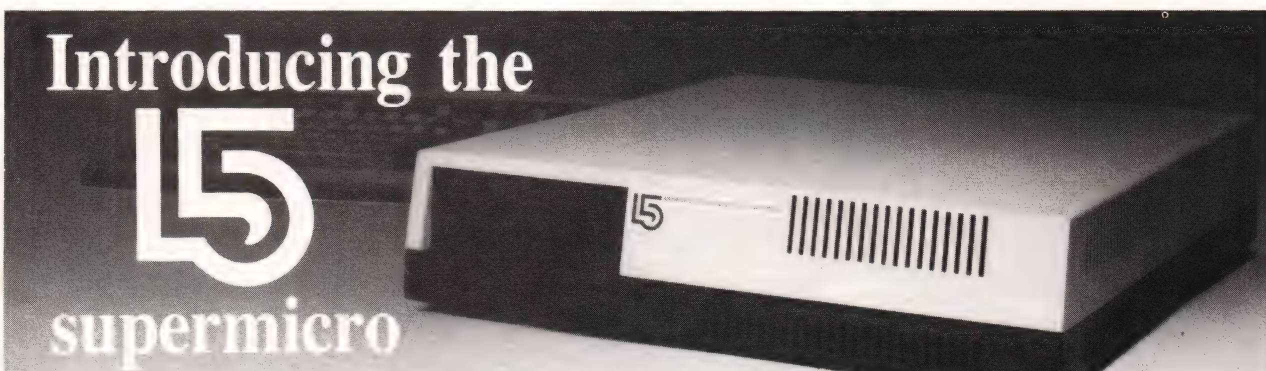
ASK ABOUT OUR "TRADE UP TO LATTICE C POLICY"

After purchase, return registration cards for free subscription to the "Lattice Works" newsletter and important information about the Lattice Users Group.

Lattice, Inc.
P.O. Box 3072
Glen Ellyn, IL 60138
(312) 858-7950
TWX 910-291-2190

International Sales Offices
Belgium: Softshop. Phone: (32) 53-664875.
England: Round Hills. Phone: (0672) 54675.
Japan: Lifeboat Japan. Phone: (03) 293-2311.

Circle **no. 58** on reader service card.



A breakthrough in Price, Performance and Packaging

The new L5 proves that good things come in small packages! Measuring a compact 3.75″H × 17.5″W × 21″D, the L5 is small enough to fit on a desk-top or a laboratory workbench. Yet it's large enough to handle up to 32 users and, in some applications, outperform a VAX system. The L5 is small in price, too. A mid-range system can cost less than $1,000 per user!

Find out how the new L5 offers the unique solution to price, performance & placement challenges. Call General Communications today!

L5 Features

- KDJ11 Processor, floating point, 8K cache
- .5Mb to 32Mb memory
- 4 to 32 users
- 20Mb to 2.5Gb external storage
- UNIX System V Fast Kernel or Real Time Kernel
- Runs RT11, RSX, and TSX
- Includes several utilities packages, necessary cabling, complete documentation and tutorials

Aggressive Dealer/OEM discounts available

*UNIX is a trademark of AT&T-Bell Laboratories
VAX, RT11, & RSX are trademarks of Digital Equipment Corporation
TSX is a trademark of S&H Computer

General Communications Corporation
"Where lucid communications, technical excellence and common sense meet."

1 Main Street, Suite 502, Eatontown, NJ 07724 (201) 542-6560

Circle **no. 13** on reader service card.

Listing Four

```
NORSTR   C744    NOSCAL   0347    ORIGIN   C5EA    PLDONE   C721
PLTSHD   C20F    PROD     00AE    PTPLOT   C628    PTPLT2   C65D
QUOT     00FD    R0       0389    RADCND   00AC    RADIUS   0377
RAM      036A    RANDOM   C0C8    RC       038D    RHEMI    C6E8
RI       038F    RNDM     C000    RO       038E    ROOT     033C
RS       038A    RT       038C    SDIV     C5EA    SPHERE   C7C7
SPOOL    CA3B    SQR      00AE    SQRT     C064    SQUARE   C004
TEMP     0384    TNTMP    037B    TONE     0379    TOROID   C90F
TPARM    C79A    VALUE    0344    XCENT    036A    XMAX     0392
XPLT     033F    XREL     036C    XSHD     036D    XSQR     0390
YCENT    036F    YPLT     0341    YREL     0370    YSHD     0371
ZEROOT   C791    ZREL     0373    ZWX      0375

END OF ASSEMBLY
```

Listing Five

```
0000                  ; INTERFACE - EASY PARAMETER SETTING FOR SHAPE
0000                  ; DRAWING ROUTINES FROM BASIC.
0000                  ;
0000                  ; RICHARD L. RYLANDER  11/23/84
0000                  ;
0000                  ;****************************************
0000                  ORIGIN=$CAC8
0000                  RAM    =$0393
0000                  ;
0000                  ; PARAMETER LOCATIONS FOR VARIOUS SHAPES
0000                  ;
0000                  XCENT  =$036A
0000                  YCENT  =$036F
0000                  XPLOT  =$033F
0000                  YPLOT  =$0341
0000                  XMIN   =$034A
0000                  YMIN   =$034C
0000                  XMID   =$034D
0000                  YMID   =$034F
0000                  XMAX   =$0350
0000                  YMAX   =$0352
0000                  RADIUS =$0377
0000                  HLEN   =$0389
0000                  RI     =$038F
0000                  RO     =$038E
0000                  ;
0000                  HVFLAG =$0383
0000                  VALUE  =$0344
0000                  PLTFLG =$033E
0000                  ;
0000                  DEFLAG =$FB
0000                  ;
0000                  ;****************************************
0000                  ;
0000                  ; FUNCTION LOCATIONS
0000                  ;
0000                  GRFON  =$C0E2            ; SWITCH TO GRAPHICS MODE
0000                  GRFOFF =$C103            ; RETURN TO TEXT DISPLAY
0000                  ;
0000                  CLEARR=$C12C             ; CLEAR BITMAP
0000                  CLRBYT=$C135             ; CLEAR (FILL) BYTE
0000                  COLORR=$C118             ; LOAD COLOR MAP
0000                  COLBYT=$C119             ; COLOR BYTE
0000                  ;
0000                  PLOTR  =$C14B            ; POINT PLOT ROUTINE
0000                  LINER  =$C2DB            ; DRAW A LINE
0000                  FACETR =$C4E1            ; DRAW A SHADED FACET
0000                  ;
0000                  ;****************************************
0000                  ;
0000                  ; SHADED SHAPE DRAWING ROUTINES
0000                  ;
0000                  SPHERR=$C7C7             ; SPHERE
0000                  CYLNDR=$C864             ; CYLINDER
0000                  TORUSR=$C90F             ; TOP-VIEW TOROID
0000                  EDGTOR=$C88F             ; EDGE-VIEW TOROID
0000                  SPOOLR=$CA3B             ; INSIDE-VIEW TOROID
0000                  ;
0000                  ;****************************************
0000                  ;
0000                  ; BASIC ROM ROUTINES
0000                  ;
0000                  CHKCOM=$AEFD             ; CHECK FOR COMMA
0000                  EVAEXP=$AD9E             ; EVALUATE EXPRESSION
0000                  FLTFIX=$B1AA             ; CONVERT TO FIXED
0000                  ;
0000                  *=RAM
0393                  LINFAC *=*+1             ; LINE OR FACET FLAG
0394                  ;
0394                  *=ORIGIN
CAC8                  ;
CAC8                  ;****************************************
CAC8                  ;
CAC8                  ; GET PARAMETERS FROM BASIC CALLING STATEMENT
CAC8                  ; OF THE FORM:
CAC8                  ; SYS(FNCTN),PARAM1,PARAM2,PARAM3[OPT]
CAC8                  ; WHERE THE THIRD PARAMETER (FOR EXAMPLE)
CAC8                  ; MAY BE OPTIONAL (A DEFAULT VALUE IS USED
CAC8                  ; IF THE PARAMETER IS NOT SPECIFIED)
CAC8                  ;
CAC8  20 FD AE        GETNUM JSR CHKCOM        ; LOOK FOR COMMA
CACB  20 9E AD               JSR EVAEXP        ; EVALUATE EXPRESSION
CACE  20 AA B1               JSR FLTFIX        ; CHANGE TO INTEGER WITH
CAD1                  ; HIGH BYTE IN "A" AND LOW BYTE IN "Y"
CAD1  60                     RTS
CAD2                  ;
CAD2                  ; CHECK FOR ADDITIONAL (OPTIONAL) PARAMETERS
CAD2                  ;
CAD2  A9 2C           PCHECK LDA #$2C          ; "," COMMA
CAD4  A0 00                  LDY #0
CAD6  84 FB                  STY DEFLAG
CAD8  D1 7A                  CMP ($7A),Y
CADA  D0 03                  BNE NOMORE        ; NO COMMA - USE DEFAULT
CADC  4C 73 00               JMP $0073
CADF  A0 80           NOMORE LDY #$80
CAE1  84 FB                  STY DEFLAG
CAE3  60                     RTS
CAE4                  ;
CAE4                  ; GET TWO ADDITIONAL PARAMETERS FOR TOROIDS
CAE4                  ;
CAE4  20 D2 CA        GETTWO JSR PCHECK
CAE7  24 FB                  BIT DEFLAG
CAE9  30 0F                  BMI DFAULT
CAEB  20 9E AD               JSR EVAEXP
CAEE  20 AA B1               JSR FLTFIX
CAF1  8C 8F 03               STY RI
CAF4  20 C8 CA               JSR GETNUM
CAF7  8C 8E 03               STY RO
CAFA  60              DFAULT RTS
CAFB                  ;
CAFB                  ;****************************************
CAFB                  ;
CAFB                  ; SET CENTER COORDINATES
CAFB                  ;
CAFB  20 C8 CA        CENTER JSR GETNUM
CAFE  8C 6A 03               STY XCENT
CB01  8D 6B 03               STA XCENT+1
CB04  20 C8 CA               JSR GETNUM
CB07  8C 6F 03               STY YCENT
CB0A  60                     RTS
CB0B                  ;
CB0B                  ;****************************************
CB0B                  ;
CB0B                  ; CLEAR THE BITMAP, FILLING WITH (OPTIONAL)
CB0B                  ; FILL VALUE SPECIFIED OR WITH (DEFAULT) "0"
CB0B                  ;
CB0B  20 D2 CA        CLEAR2 JSR PCHECK
CB0E  24 FB                  BIT DEFLAG
CB10  30 07                  BMI DEFCLR
CB12  20 9E AD               JSR EVAEXP
CB15  20 AA B1               JSR FLTFIX
CB18  2C                     .BYTE $2C
CB19  A0 00           DEFCLR LDY #0
CB1B  8C 35 C1               STY CLRBYT
CB1E  4C 2C C1               JMP CLEARR
CB21                  ;
CB21                  ;****************************************
CB21                  ;
CB21                  ; FILL COLOR MAP WITH (OPTIONAL) COLOR BYTE
CB21                  ; SPECIFIED OR WITH (DEFAULT) "$01"
CB21                  ; (BLACK DOTS ON WHITE BACKGROUND)
CB21                  ;
CB21  20 D2 CA        COLOR2 JSR PCHECK
CB24  24 FB                  BIT DEFLAG
CB26  30 07                  BMI DEFCOL
CB28  20 9E AD               JSR EVAEXP
CB2B  20 AA B1               JSR FLTFIX
CB2E  2C                     .BYTE $2C
CB2F  A0 01           DEFCOL LDY #$01
CB31  8C 19 C1               STY COLBYT
CB34  4C 18 C1               JMP COLORR
CB37                  ;
CB37                  ;****************************************
CB37                  ;
CB37                  ; PLOT OR UNPLOT POINTS
CB37                  ;
CB37  A9 00           PLOT2  LDA #0
CB39  2C                     .BYTE $2C
CB3A  A9 80           UNPLT2 LDA #$80
CB3C  8D 3E 03               STA PLTFLG
CB3F  20 C8 CA               JSR GETNUM
CB42  8C 3F 03               STY XPLOT
CB45  8D 40 03               STA XPLOT+1
CB48  20 C8 CA               JSR GETNUM
CB4B  8C 41 03               STY YPLOT
CB4E  4C 4B C1               JMP PLOTR
CB51                  ;
CB51                  ;****************************************
CB51                  ;
CB51                  ; DRAW LINES BETWEEN (X1,Y1) AND (X2,Y2)
CB51                  ; OR SHADED FACETS BETWEEN THREE POINTS
CB51                  ; (X1,Y1), (X2,Y2) AND (X3,Y3)
CB51                  ;
CB51  A9 00           LINE2  LDA #0
CB53  2C                     .BYTE $2C
CB54  A9 80           FACET2 LDA #$80
CB56  8D 93 03               STA LINFAC
CB59  20 C8 CA               JSR GETNUM
CB5C  8C 4A 03               STY XMIN
CB5F  8D 4B 03               STA XMIN+1
CB62  20 C8 CA               JSR GETNUM
CB65  8C 4C 03               STY YMIN
CB68  20 C8 CA               JSR GETNUM
CB6B  8C 4D 03               STY XMID
CB6E  8D 4E 03               STA XMID+1
CB71  20 C8 CA               JSR GETNUM
CB74  8C 4F 03               STY YMID
CB77  2C 93 03               BIT LINFAC
CB7A  10 18                  BPL LDRAW
CB7C  20 C8 CA               JSR GETNUM
CB7F  8C 50 03               STY XMAX
CB82  8D 51 03               STA XMAX+1
CB85  20 C8 CA               JSR GETNUM
CB88  8C 52 03               STY YMAX
CB8B  20 C8 CA               JSR GETNUM
CB8E  8C 44 03               STY VALUE
CB91  4C E1 C4               JMP FACETR
CB94  4C DB C2        LDRAW  JMP LINER
CB97                  ;
CB97                  ;****************************************
CB97                  ;
CB97                  ; DRAW A SPHERE CENTERED AT (XCENT,YCENT)
CB97                  ; DEFAULT RADIUS IS LAST VALUE USED
CB97                  ;
CB97  20 FB CA        SPHER2 JSR CENTER
CB9A  20 D2 CA               JSR PCHECK
CB9D  24 FB                  BIT DEFLAG
CB9F  30 09                  BMI SKIP1
CBA1  20 9E AD               JSR EVAEXP
CBA4  20 AA B1               JSR FLTFIX
CBA7  8C 77 03               STY RADIUS
CBAA  4C C7 C7        SKIP1  JMP SPHERR
CBAD                  ;
CBAD                  ;****************************************
CBAD                  ;
CBAD                  ; DRAW A TOP-VIEW TOROID AT (XCENT,YCENT)
CBAD                  ; DEFAULT INNER AND OUTER RADII ARE LAST USED
CBAD                  ;
CBAD  20 FB CA        TORUS2 JSR CENTER
CBB0  20 E4 CA               JSR GETTWO
CBB3  4C 0F C9               JMP TORUSR
CBB6                  ;
CBB6                  ;****************************************
CBB6                  ;
CBB6                  ; DRAW CYLINDERS WITH AXES HORIZONTAL OR
CBB6                  ; VERTICAL.  DEFAULT RADIUS AND "HALF-LENGTH"
CBB6                  ; ARE LAST VALUES USED.
CBB6                  ;
CBB6  A9 80           VCYL2  LDA #$80
CBB8  2C                     .BYTE $2C
CBB9  A9 00           HCYL2  LDA #0
CBBB  8D 83 03               STA HVFLAG
CBBE  20 FB CA               JSR CENTER
CBC1  20 D2 CA               JSR PCHECK
CBC4  24 FB                  BIT DEFLAG
CBC6  30 0F                  BMI SKIP2
CBC8  20 9E AD               JSR EVAEXP
```

(Continued on page 78)

Does your ISAM run on IBM, APPLE, DEC and AT&T computers? c-tree does, and you only BUY IT ONCE!

c-tree™ BY FAIRCOM

2606 Johnson Drive
Columbia MO 65203

The company that introduced micros to B+ Trees in 1979 and created ACCESS MANAGER™ for Digital Research, now redefines the market for high performance, B+ Tree based file handlers. With c-tree™ you get:

- complete C source code written to K&R standards of portability
- high level, multi-key ISAM routines and low level B+ Tree functions
- routines that work with single-user and network systems
- no royalties on application programs

$395 COMPLETE

Specify format:
5¼" PC-DOS 3½" Mac
8" CP/M® 8" RT-11

for VISA, MC or COD orders, call
1-314-445-6833

Access Manager and CP/M are trademarks of Digital Research, Inc. Apple is a trademark of Apple Computer, Inc. c-tree and the circular disc logo are trademarks of FairCom IBM is a trademark of International Business Machines Corporation DEC is a trademark of Digital Equipment Corporation ©1984 FairCom

Circle **no. 37** on reader service card.

$5.00 C Compiler

Due to popular demand, **Dr. Dobb's Journal** has reprinted its most-asked-for C compiler articles by Ron Cain and J. E. Hendrix, each for only $5.00.

Ron Cain's C compiler from sold-out 1980 issues #45 and #48 includes "A Small C Compiler for the 8080s" and "Runtime Library for the Small C Compiler."

The J. E. Hendrix reprint includes part two of "Small-C Compiler v.2" from sold out issue #75 and completes the first part of the compiler article from issue #74 which is included in Dr. Dobb's Bound Volume 7.

To Order: Enclose $5.00 for each copy with this coupon and send to:
Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303
Outside U.S., add $2.00 per copy for shipping and handling.

Please send _____ copy(ies) of the Ron Cain Reprint, and
_____ copy(ies) of the J. E. Hendrix reprint to:

Name ______________________________

Address ______________________________

City ____________________ State _____ Zip ________

ALL REPRINT ORDERS MUST BE PREPAID.

Please allow 6–9 weeks for delivery.

C Programmers: Program three times faster with *Instant-C*™

Instant-C™ is an optimizing interpreter for the C language that can make programming in C three or more times faster than using old-fashioned compilers and loaders. The interpreter environment makes C as easy to use as Basic. Yet ***Instant-C***™ is 20 to 50 times faster than interpreted Basic. This new interactive development environment gives you:

Instant Editing. The full-screen editor is built into ***Instant-C***™ for immediate use. You don't wait for a separate editor program to start up.

Instant Error Correction. You can check syntax in the editor. Each error message is displayed on the screen with the cursor set to the trouble spot, ready for your correction. Errors are reported clearly, by the editor, and only one at a time.

Instant Execution. ***Instant-C***™ uses no assembler or loader. You can execute your program as soon as you finish editing.

Instant Testing. You can immediately execute any C statement or function, set variables, or evaluate expressions. Your results are displayed automatically.

Instant Symbolic Debugging. Watch execution by single statement stepping. Debugging features are built-in; you don't need to recompile or reload using special options.

Instant Loading. Directly generates .EXE or .CMD files at your request to create stand-alone versions of your programs.

Instant Floating Point. Uses 8087* co-processor if present.

Instant Compatibility. Follows K & R standards. Comprehensive standard library provided, with source code.

Instant Satisfaction. Guaranteed, or your money back. ***Instant-C***™ is available now, and works under PC-DOS, MS-DOS*, and CP/M-86*.

Find out how ***Instant-C***™ is changing the way that programming is done. ***Instant-C***™ is $495. Call or write for more information.

Rational Systems, Inc.
(617) 653-6194
P.O. Box 480
Natick, Mass. 01760

Trademarks: MS-DOS (Microsoft Corp.), 8087 (Intel Corp.), CP/M-86 (Digital Research, Inc.), Instant-C (Rational Systems, Inc.)

Circle **no. 79** on reader service card.

Drawing on the C-64 (Listing Continued, text begins on page 50)

Listing Five

```
CBCB  20 AA B1        JSR FLTFIX
CBCE  8C 77 03        STY RADIUS
CBD1  20 C8 CA        JSR GETNUM
CBD4  8C 89 03        STY HLEN
CBD7  4C 64 C8  SKIP2 JMP CYLNDR
CBDA                  ;
CBDA                  ;****************************************
CBDA                  ;
CBDA                  ; DRAW EDGE-VIEW TOROIDS WITH AXES HORIZONTAL
CBDA                  ; OR VERTICAL
CBDA                  ; INNNER AND OUTER RADII ARE OPTIONAL
CBDA                  ;
CBDA  A9 80     VTOR2 LDA ##80
CBDC  2C              .BYTE #2C
CBDD  A9 00     HTOR2 LDA #0
CBDF  8D 83 03        STA HVFLAG
CBE2  20 FB CA        JSR CENTER
CBE5  20 E4 CA        JSR GETTWO
CBE8  4C 8F C8        JMP EDGTOR
CBEB                  ;
CBEB                  ;****************************************
CBEB                  ;
CBEB                  ; DRAW INSIDE-VIEW TOROIDS, "SPOOLS",
CBEB                  ; WITH AXES HORIZONTAL OR VERTICAL
CBEB                  ; INNER AND OUTER RADII ARE OPTIONAL
CBEB                  ;
CBEB  A9 80     VSPL2 LDA ##80
CBED  2C              .BYTE #2C
CBEE  A9 00     HSPL2 LDA #0
CBF0  8D 83 03        STA HVFLAG
CBF3  20 FB CA        JSR CENTER
CBF6  20 E4 CA        JSR GETTWO
CBF9  4C 3B CA        JMP SPOOLR
CBFC                  .END

ERRORS = 00000

SYMBOL TABLE

SYMBOL VALUE
 CENTER   CAFB    CHKCOM   AEFD    CLEAR2   CB0B    CLEARR   C12C
 CLRBYT   C135    COLBYT   C119    COLOR2   CB21    COLORR   C118
 CYLNDR   C864    DEFCLR   CB19    DEFCOL   CB2F    DEFLAG   00FB
 DFAULT   CAFA    EDGTOR   C88F    EVAEXP   AD9E    FACET2   CB54
 FACETR   C4E1    FLTFIX   B1AA    GETNUM   CAC8    GETTWO   CAE4
 GRFOFF   C103    GRFON    C0E2    HCYL2    CBB9    HLEN     0389
 HSPL2    CBEE    HTOR2    CBDD    HVFLAG   0383    LDRAW    CB94
 LINE2    CB51    LINER    C2DB    LINFAC   0393    NOMORE   CADF
 ORIGIN   CACB    PCHECK   CAD2    PLOT2    CB37    PLOTR    C14B
 PLTFLG   033E    RADIUS   0377    RAM      0393    RI       038F
 RO       038E    SKIP1    CBAA    SKIP2    CBD7    SPHER2   CB97
 SPHERR   C7C7    SPOOLR   CA3B    TORUS2   CBAD    TORUSR   C90F
 UNPLT2   CB3A    VALUE    0344    VCYL2    CBB6    VSPL2    CBEB
 VTOR2    CBDA    XCENT    036A    XMAX     0350    XMID     034D
 XMIN     034A    XPLOT    033F    YCENT    036F    YMAX     0352
 YMID     034F    YMIN     034C    YPLOT    0341

END OF ASSEMBLY
```

End Listing Five

Listing Six

```
10 REM        SHAPES DEMO
20 :
30 REM     RICHARD L. RYLANDER    11/23/84 (REVISED 1/20/85 TO ADD LABELING)
40 :
50 GR=49378     :REM GRAPHICS MODE
60 TX=49411     :REM TEXT MODE
70 :
80 LB=893       :REM LEFT BOUND
90 RB=894       :REM RIGHT BOUND
100 UB=895      :REM UP BOUND
110 DB=896      :REM DOWN BOUND
120 :
130 REM   FLAGS FOR VARIOUS DRAWING MODES
140 :
150 SH=838      :REM SHADE STYLE - 0=RANDOM, 1=HALFTONE
160 SC=839      :REM SCALING - 0=NORMAL (1:1), 1=SCALED (3:4) FOR SCREEN DISPLAY
170 LT=898      :REM LIGHTING - 0=NORMAL SINGLE-SOURCE, 1=BACKLIT ILLUMINATION
180 :
190 BO=53280    :REM  BORDER COLOR
200 :
210 REM   FUNCTION LOCATIONS
220 :
230 CL=51979    :REM CLEAR BITMAP AREA
240 CO=52001    :REM FILL COLOR MAP
250 :
260 SP=52119    :REM  SPHERE
270 TR=52141    :REM  TOP-VIEW TOROID
280 VC=52150    :REM  CYLINDER (AXIS VERTICAL)
290 HC=52153    :REM  CYLINDER (AXIS HORIZONTAL)
300 VT=52186    :REM  EDGE-VIEW TOROID (AXIS VERTICAL)
310 HT=52189    :REM  EDGE-VIEW TOROID (AXIS HORIZONTAL)
320 VS=52203    :REM  INSIDE-VIEW TOROID ["SPOOL"] (AXIS VERTICAL)
330 HS=52206    :REM  INSIDE-VIEW TOROID ["SPOOL"] (AXIS HORIZONTAL)
340 :
350 REM  DRAW SAMPLES OF PRIMITIVE SHAPES WITH LABEL BENEATH EACH
360 :
370 POKE SH,1 :REM HALFTONE SHADING
380 POKE SC,1 :REM USE SCREEN SCALING (FOR UNDISTORTED SPHERES, ETC.)
390 POKE LT,0 :REM USE NORMAL (NO BACKLIGHT) ILLUMINATION
400 SYS(CL)    :REM CLEAR SCREEN (DEFAULT - FILL BITMAP WITH 0'S)
410 SYS(CO),16*11+1  :REM COLOR COMBINATION - DARK GRAY (11) DOTS ON WHITE (1)
412 REM     LOOKS BETTER THAN DEFAULT COLOR COMBINATION OF BLACK DOTS ON WHITE
414 REM     ON MOST COLOR MONITORS
420 POKE BO,1 :REM WHITE BORDER TO MATCH BACKGROUND
430 SYS(GR)    :REM ENTER BITMAPPED GRAPHICS MODE
432 RW=12:CM=14:MD=1:A$="SAMPLE SHAPES":GOSUB 1900:REM  PUT TITLE ON IMAGE
434 X1=110:Y1=120:X2=210:Y2=120:BC=1:DC=7:GOSUB 1700:REM  COLOR TITLE TEXT
440 POKE LB,38:POKE RB,38:POKE UB,38:POKE DB,38        :REM INITIALIZE CLIPPING
450 SYS(SP),40,199,38
455 RW=9:CM=2:MD=1:A$="SPHERE":GOSUB1900
460 SYS(HC),120,199,38,38
465 CM=11:A$="H-CYLNDR":GOSUB1900
470 SYS(VC),200,199 :REM NO SIZE PARAMETERS GIVEN, DEFAULT (PREVIOUS) ARE USED
475 CM=21:A$="V-CYLNDR":GOSUB1900
480 SYS(TR),280,199,15,38
485 CM=32:A$="TOROID":GOSUB1900
490 SYS(VT),40,64
495 RW=23:CM=1:A$="V-TOROID":GOSUB1900
500 SYS(HT),120,64
505 CM=11:A$="H-TOROID":GOSUB1900
510 SYS(HS),200,64,5,100
515 CM=22:A$="H-SPOOL":GOSUB1900
520 SYS(VS),280,64
525 CM=32:A$="V-SPOOL":GOSUB1900
530 POKE 198,0:WAIT 198,1:POKE 198,0
540 REM  WAIT FOR A KEY TO BE PRESSED
550 :
560 REM  DRAW TWO "GOBLETS" ONE WITH HALFTONE, THE OTHER RANDOM SHADING.
570 :
580 SYS(CL):SYS(CO),16*11+1
582 RW=14:CM=15:A$="COMPARISON":GOSUB 1900:RW=15:CM=19:A$="OF":GOSUB 1900
584 RW=16:CM=14:A$="SHADE STYLES":GOSUB 1900:RW=18:CM=14:A$="<-- HALFTONE"
586 GOSUB 1900:RW=20:CM=15:A$="RANDOM -->":GOSUB 1900
590 POKE LB,255:POKE RB,255:POKE UB,49:POKE DB,255: REM CLIP AT SCREEN TOP
600 SYS(SP),80,190,80
610 POKE UB,51:POKE DB,51: REM CLIP AT JUNCTION WITH SPHERE FOR SMOOTH SEAM
620 SYS(VS),80,69,10,130
630 POKE DB,9:POKE UB,8
640 SYS(VT),80,9,25,45
650 POKE SH,0  :REM SWITCH TO RANDOM SHADING
660 POKE LB,255:POKE RB,255:POKE UB,49:POKE DB,255
670 SYS(SP),240,190,80
680 POKE UB,51:POKE DB,51
690 SYS(VS),240,69,10,130
700 POKE DB,9:POKE UB,8
710 SYS(VT),240,9,25,45
720 POKE 198,0:WAIT 198,1:POKE 198,0
740 :
750 REM  DRAW "WINE" SCENE
760 POKE LT,1 :REM BACKLIT ILLUMINATION
770 POKE SH,1 :REM HALFTONE SHADING FOR MOST (RANDOM "LABEL" ON BOTTLE)
780 SYS(CO):SYS(CL),255: REM FILL BITMAP WITH 1'S ("SET" BACKGROUND)
790 POKE BO,0 : REM BLACK BORDER TO MATCH BACKGROUND
792 RW=0:CM=0:MD=2:A$="DRAWING  WITH":GOSUB 1900:RW=1:CM=2:A$="BACKLIGHT"
794 GOSUB 1900:RW=2:CM=0:A$="AGAINST A SET":GOSUB 1900
796 RW=3:CM=2:A$="BACKGROUND":GOSUB 1900
798 RW=1:CM=26:A$="COLORS  ADDED":GOSUB 1900
800 RW=2:CM=25:A$="TO SELECT AREAS":GOSUB 1900
810 REM  DRAW BOTTLE
820 POKE UB,0:POKE DB,255:POKE LB,255:POKE RB,255
830 SYS(VT),150,10,30,50
840 POKE UB,255:SYS(VC),150,70,50,60
850 POKE DB,0:SYS(VT),150,130,6,50
860 POKE DB,55:POKE UB,0:SYS(VS),150,204,15,181
870 POKE UB,255:SYS(VC),150,221,16,17
880 :
890 REM  DRAW WINE GLASS
900 POKE UB,20:SYS(SP),80,120,60
910 POKE UB,35:POKE DB,34:SYS(VS),80,34,10,110
920 :
930 REM  DRAW SOME GRAPES
940 SYS(SP),8,8,8: REM OTHER "GRAPES" WILL BE SAME RADIUS - JUST GIVE POSITIONS
950 SYS(SP),20,8:SYS(SP),40,8:SYS(SP),12,20:SYS(SP),30,20:SYS(SP),25,16
960 :
970 REM  DRAW AN APPLE BY A PAIR OF EDGE-VIEW TOROIDS AND A SPHERE SECTION
980 POKE UB,255:POKE DB,255:POKE LB,255:POKE RB,59
990 SYS(VT),260,29,0,50:SYS(VT),260,79
1000 POKE UB,43:POKE DB,43:SYS(SP),260,54,60
1010 REM   PUT STEM ON APPLE
1020 POKE RB,0:POKE DB,0:SYS(TR),272,104,10,15
1030 REM   ADD A LEAF BY A SPHERE SECTION
1040 POKE DB,255:POKE RB,0:SYS(SP),256,119,15
1050 REM  ADD A RANDOM SHADED "LABEL" TO THE BOTTLE
1060 POKE UB,255:POKE RB,255:POKE LB,6
1070 POKE,SH,0:SYS(VC),150,72,50,48
1080 :
1090 REM  ADD COLORS TO "WINE" SCENE
1100 SYS(CO),12:REM REPLACE WHITE "HOLES" (BACKGROUND) WITH MEDIUM GRAY
1102 X1=0:Y1=200:X2=100:Y2=239:DC=0:BC=0:GOSUB 1700:REM HIDE TEXT IN CORNERS
1104 X1=180:Y1=200:X2=319:Y2=239:DC=0:BC=0:GOSUB 1700
1110 X1=200:Y1=1:X2=315:Y2=100:DC=0:BC=2:GOSUB 1700
1120 X1=240:Y1=110:X2=255:Y2=150:BC=5:GOSUB 1700
1130 X1=260:Y1=110:X2=270:Y2=135:BC=9:GOSUB 1700
1140 X1=1:Y1=1:X2=48:Y2=30:BC=4:GOSUB 1700
1150 X1=140:Y1=205:X2=180:Y2=235:BC=7:GOSUB 1700
1160 X1=145:Y1=25:X2=195:Y2=115:BC=6:GOSUB 1700
1170 POKE 198,0:WAIT 198,1:POKE 198,0
1190 :
1200 REM   "COFFEE AND DONUTS"
1210 POKE SH,0 :REM RANDOM SHADING ON DONUTS
1220 SYS(CO),16*11+1:SYS(CL):POKE BO,1:REM WHITE BORDER TO MATCH BACKGROUND
1230 POKE LB,255:POKE RB,255:POKE UB,255:POKE DB,255 :REM NO INITIAL CLIPPING
1240 SYS(VT),60,20,20,60
1250 POKE RB,29:SYS(VT),99,60:POKE RB,255
1260 SYS(TR),188,180
1270 REM ADD COFFEE CUP (HALFTONE SHADING)
1280 POKE SH,1:POKE UB,0:SYS(VT),188,20:POKE UB,255
1290 POKE DB,0:SYS(TR),278,110,20,40
1300 POKE DB,255:POKE UB,0:POKE LB,0
1310 SYS(TR),248,90,50,70:POKE LB,255:SYS(SP),248,110,10:POKE UB,255
1320 SYS(VC),308,100,10,10
1330 SYS(VC),188,77,60,57
1340 POKE DB,0:SYS(VT),188,134,40,60
1350 REM  ADD COLOR - FIRST MAKE SCREEN BROWN DOTS ON WHITE THEN MAKE CUP GREEN
1360 SYS(CO),1+16*9 :REM 1=WHITE BACKGROUND, 9=BROWN DOTS
1370 X1=130:Y1=1:X2=319:Y2=136:BC=1:DC=5:GOSUB 1700
1380 X1=250:Y1=144:X2=319:Y2=144:GOSUB 1700
1390 POKE 198,0:WAIT 198,1:POKE 198,0
1410 :
1420 REM  DRAW "LINKED" TOROIDS BY REDRAWING OVERLAP AREAS USING CLIPPING
1430 POKE LT,0 :REM BLUE DOTS ON WHITE, NO BACKLIGHT
1440 SYS(CL):SYS(CO),1+16*6: REM 1=WHITE BACKGROUND, 6=BLUE DOTS
1450 POKE UB,255:POKE DB,255:POKE LB,255:POKE RB,255: REM NO INITIAL CLIPPING
1460 POKE SH,0  :REM RANDOM SHADING
1470 SYS(TR),244,84,48,70
1480 SYS(TR),160,84:SYS(TR),76,84
1490 SYS(TR),118,156:SYS(TR),202,156
1500 REM  ADD PROPER OVERLAP SECTIONS
1510 POKE RB,0:POKE DB,0:SYS(TR),160,84:POKE RB,255:POKE LB,0
1520 SYS(TR),76,84:POKE DB,255:POKE UB,0:SYS(TR),118,156
1530 POKE LB,255:POKE RB,0:SYS(TR),202,156:POKE LB,27
1540 POKE DB,0:POKE UB,255:SYS(TR),160,84
1550 POKE DB,0:POKE UB,255:SYS(TR),160,84
1560 POKE LB,255:POKE UB,27:SYS(TR),244,84
1570 POKE LB,0:POKE RB,27:POKE UB,255:SYS(TR),244,84
1580 POKE 198,0
1590 GET A$:IF A$= "" THEN 1590
1600 SYS(TX):POKE BO,14:REM RETURN TO TEXT MODE
1610 END
1620 :
1630 REM SUBROUTINE FOR ADDING COLOR TO DIFFERENT SCREEN AREAS
1640 REM REMEMBER THAT COLOR BOUNDARIES MUST CORRESPOND TO CHARACTER BOUNDARIES
1650 REM DEFINE A RECTANGULAR AREA BY LOWER-LEFT AND UPPER-RIGHT COORDINATES
1660 REM (X1,Y1)=LOWER-LEFT POINT, (X2,Y2)=UPPER-RIGHT POINT
1670 REM THE CORNER POINTS CAN BE ARBITRARY BUT COLOR CHANGE WILL BE IN THE
1680 REM SMALLEST CHARACTER-CELL BOUNDED RECTANGLE THAT INCLUDES THOSE POINTS
1690 REM  Y-COORDINATES MUST BE "UNSCALED" IF SCALE FLAG IS SET:
1700 IF PEEK(SC)THEN Y1=(Y1+1)*213/256:Y2=(Y2+1)*213/256
1710 REM COLOR TO BE POKED IN IS GIVEN BY VARIABLE CC="COMPOSITE COLOR"
1720 REM WHERE CC=16*DC + BC [DC=DOT COLOR, BC=BACKGROUND COLOR NUMBERS]
1730 CC=16*DC+BC
1740 FOR IX=INT(X1/8) TO INT(X2/8)
1750 FOR IY=INT(Y1/8) TO INT(Y2/8)
```

(Continued on page 80)

Drawing on the C-64 (Listing Continued, text begins on page 50)

Listing Six

```
1760 POKE 34752+IX-40*IY,CC
1770 NEXT:NEXT:RETURN
1780 :
1790 REM SUBROUTINE TO ADD TEXT TO GRAPHIC SCREEN.
1800 REM "RW" AND "CM" ARE THE ROW (0-24) AND COLUMN (0-39) COORDINATES FOR THE
1810 REM FIRST LETTER OF THE TEXT STRING TO BE PRINTED.
1820 REM THE TEXT STRING ITSELF IS ASSIGNED TO "A$".
1830 REM "MD" IS THE MODE FOR THE PRINTING. FIVE MODES ARE ALLOWED:
1840 REM  1 - NORMAL ("BLACK" LETTERS ON "WHITE" BACKGROUND)
1850 REM  2 - REVERSED ("WHITE" LETTERS ON "BLACK" BACKGROUND)
1860 REM  3 - SET ("BLACK") LETTERS "OR'ED" WITH BACKGROUND
1870 REM  4 - UNSET ("WHITE") LETTERS "AND'ED" WITH BACKGROUND
1880 REM  5 - SET LETTERS "XOR'ED" WITH BACKGROUND
1890 :
1900 SB=40952:TB=54272:IF(MD AND 1)THEN TB=53248:REM SCREEN AND TEXT BASE ADDR'S
1910 OS=320*RW+8*CM:REM  OFFSET FROM CHARACTER SCREEN BASE
1920 POKE 56334,PEEK(56334)AND 254:REM DISABLE IRQ TIMER
1930 POKE 1,PEEK(1)AND 251:REM SWITCH CHARACTER ROM IN
1940 L=LEN(A$):FOR N=1 TO L:N8=N*8+OS+SB
1950 X=ASC(MID$(A$,N,1)):IF X>63 THEN X=X-64
1960 TC=TB+8*X
1970 ON MD GOTO 1980,1980,1990,2000,2010
1980 POKE 53231,36:GOTO 2020
1990 POKE 53231,17:GOTO 2020
2000 POKE 53231,49:GOTO 2020
2010 POKE 53231,81
2020 POKE252,N8/256:POKE251,N8-256*INT(N8/256)
2030 POKE254,TC/256:POKE253,TC-256*INT(TC/256)
2040 SYS(53221):NEXT
2050 POKE 1,PEEK(1)OR 4:POKE 56334,PEEK(56334)OR 1:REM RESTORE TO NORMAL
2060 RETURN

READY.
```

End Listing Six

Listing Seven

```
10 REM "STELLATION" DRAW A SMALL STELLATED DODECAHEDRON IN VARIOUS ORIENTATIONS
20 REM  AND STYLES   RICHARD L. RYLANDER      12/5/84
30 :
40 GR=49378    :REM GRAPHICS MODE
50 TX=49411    :REM TEXT MODE
60 BO=53280    :REM BORDER COLOR
70 :
80 REM  INITIALIZE A FEW STYLE PARAMETERS
90 POKE 839,1  :REM SCALE (3:4) FOR UNDISTORTED SCREEN DISPLAY
100 POKE 871,0 :REM FACET EDGE/LINE MODE (0=DRAW, 1=ERASE)
110 SH=838     :REM SHADE STYLE (0=RANDOM, 1=HALFTONE)
120 EG=868     :REM EDGES FLAG (0=NORMAL, 1=ADD LINES TO FACET EDGES)
130 :
140 REM   FUNCTION LOCATIONS
150 CL=51979   :REM CLEAR BITMAP
160 CO=52001   :REM FILL COLOR MAP
170 FC=52052   :REM DRAW A SHADED FACET
180 KS=53081   :REM DO KEYED SORT
190 :
200 XC=160:YC=120  :REM (SCALED) SCREEN CENTER COORDINATES
210 :
220 PRINT"(SC)    **********************************"
230 PRINT"    * SMALL STELLATED DODECAHEDRON *"
240 PRINT"    **********************************"
250 PRINT"(CD)(CD) SELECT SHADING STYLE:"
260 PRINT" R=RANDOM, H=HALFTONE"
270 INPUT"(CD) YOUR CHOICE   H(CL)(CL)(CL)";A$
280 POKE SH,0:IF A$="H" THEN POKE SH,1
290 PRINT"(CD)(CD) SELECT STYLE FOR DRAWING:"
300 PRINT"(CD) N - NORMAL":PRINT"(CD) E - EDGES EMPHASIZED"
310 PRINT"(CD) W - WIRE FRAME (HIDDEN LINES REMOVED)"
320 INPUT"(CD) YOUR CHOICE   N(CL)(CL)(CL)";A$
330 POKE EG,0:WI=0:IF A$="N" THEN 360
340 POKE EG,1:IF A$="W" THEN WI=-1
350 :
360 PRINT"(CD) READING VERTEX DATA"
370 VN=32:DIM P%(VN-1,2):REM  VN = NUMBER OF VERTICES
380 FOR N=0 TO VN-1:READ P%(N,0),P%(N,1),P%(N,2):NEXT
390 :
400 PRINT"(CD) ENTER X, Y, AND Z ANGLES FOR ROTATION"
410 PRINT"        (ANGLES IN DEGREES)"
420 INPUT X,Y,Z
430 J=3.14159265/180:X=X*J:Y=Y*J:Z=Z*J
440 X0=COS(Y)*COS(Z)-SIN(X)*SIN(Y)*SIN(Z):X1=COS(Y)*SIN(Z)+SIN(X)*SIN(Y)*COS(Z)
450 X2=-COS(X)*SIN(Y):Y0=-COS(X)*SIN(Z):Y1=COS(X)*COS(A):Y2=SIN(X)
460 Z0=SIN(Y)*COS(Z)+SIN(X)*COS(Y)*SIN(Z)
470 Z1=SIN(Y)*SIN(Z)-SIN(X)*COS(Y)*COS(Z):Z2=COS(X)*COS(Y)
480 PRINT"(CD) PERFORMING ROTATION"
490 FOR N=0 TO VN-1
500 X=P%(N,0):Y=P%(N,1):Z=P%(N,2)
510 P%(N,0)=X0*X+X1*Y+X2*Z:P%(N,1)=Y0*X+Y1*Y+Y2*Z:P%(N,2)=Z0*X+Z1*Y+Z2*Z:NEXT
520 :
530 FA=60: REM  TOTAL NUMBER OF FACETS
540 DIM F%(FA/2,2),SH(FA/2),Z%(FA/2),K%(FA/2)
550 PRINT" READING CONNECTION DATA                 "
560 VF=-1: REM  VF = # VISIBLE FACETS
570 FOR N=1 TO FA
580 VF=VF+1
590 FOR I=0 TO 2 :READ F%(VF,I):NEXT
600 REM  CALCULATE COMPONENTS OF NORMAL VECTOR TO DETERMINE VISIBILITY
610 Z=(P%(F%(VF,2),0)-P%(F%(VF,1),0))*(P%(F%(VF,0),1)-P%(F%(VF,1),1))
620 Z=Z-(P%(F%(VF,0),0)-P%(F%(VF,1),0))*(P%(F%(VF,2),1)-P%(F%(VF,1),1))
630 IF Z=<0 THEN 720:REM  FACET NOT VISIBLE, GO ON
640 X=(P%(F%(VF,2),1)-P%(F%(VF,1),1))*(P%(F%(VF,0),2)-P%(F%(VF,1),2))
650 X=X-(P%(F%(VF,0),1)-P%(F%(VF,1),1))*(P%(F%(VF,2),2)-P%(F%(VF,1),2))
660 Y=(P%(F%(VF,2),2)-P%(F%(VF,1),2))*(P%(F%(VF,0),0)-P%(F%(VF,1),0))
670 Y=Y-(P%(F%(VF,0),2)-P%(F%(VF,1),2))*(P%(F%(VF,2),0)-P%(F%(VF,1),0))
680 NC=SQR(X*X+Y*Y+Z*Z):REM  LENGTH OF NORMAL VECTOR
690 SH(VF)=26*(2*Z+X+Y)/NC
700 SH(VF)=(SH(VF)+64)*(SH(VF)+64)/256:REM RAISED COSINE SHADING
710 GOTO 730
720 VF=VF-1
730 NEXT
740 :
750 PRINT" SCALING TO DISPLAY SIZE"
760 Y=0:FOR N=0 TO VN-1:IF ABS(P%(N,1))>Y THEN Y=ABS(P%(N,1))
770 NEXT:S=119/Y
780 FOR N=0 TO VN-1:P%(N,1)=S*P%(N,1)+YC:P%(N,0)=S*P%(N,0)+XC:NEXT
790 :
800 REM FIND AVERAGE Z FOR EACH FACET
810 FOR N=0 TO VF
820 Z%(N)=(P%(F%(N,0),2)+P%(F%(N,1),2)+P%(F%(N,2),2))/3:NEXT
830 :
840 PRINT" SORTING FACETS ACCORDING TO AVG 'Z' "
850 POKE 140,VF
860 K%(0)=K%(0):POKE 251,PEEK(71):POKE 252,PEEK(72)
870 Z%(0)=Z%(0):POKE 253,PEEK(71):POKE 254,PEEK(72)
880 SYS(KS)
890 :
900 REM DRAW FACETS (ACCORDING TO Z DEPTH SINCE NOT CONVEX)
910 SYS(GR):SYS(CO):SYS(CL):POKE BO,1
920 FOR N=0 TO VF:FA=K%(N)
930 IF WI THEN SH(FA)=64
940 X0=P%(F%(FA,0),0):Y0=P%(F%(FA,0),1):X1=P%(F%(FA,1),0):Y1=P%(F%(FA,1),1)
950 X2=P%(F%(FA,2),0):Y2=P%(F%(FA,2),1)
960 SYS(FC),X0,Y0,X1,Y1,X2,Y2,SH(FA)
970 NEXT
980 POKE198,0
990 GET A$:IF A$="" THEN 990
1000 SYS(TX):POKE BO,14:END
1010 :
1020 REM      VERTEX DATA
1030 DATA 1000,618,0, 1000,-618,0, -1000,618,0, -1000,-618,0
1040 DATA 0,1000,618, 0,1000,-618, 0,-1000,618, 0,-1000,-618
1050 DATA 618,0,1000, -618,0,1000, 618,0,-1000, -618,0,-1000
1060 DATA 618,0,236, 618,0,-236, -618,0,236, -618,0,-236
1070 DATA 236,618,0, -236,618,0, 236,-618,0, -236,-618,0
1080 DATA 0,236,618, 0,-236,618, 0,236,-618, 0,-236,-618
1090 DATA 382,382,382, 382,382,-382, 382,-382,382, 382,-382,-382
1100 DATA -382,382,382, -382,382,-382, -382,-382,382, -382,-382,-382
1110 :
1120 REM    CONNECTION DATA
1130 DATA 0,12,13, 0,13,25, 0,25,16, 0,16,24, 0,24,12
1140 DATA 1,12,26, 1,26,18, 1,18,27, 1,27,13, 1,13,12
1150 DATA 2,15,14, 2,14,28, 2,28,17, 2,17,29, 2,29,15
1160 DATA 3,14,15, 3,15,31, 3,31,19, 3,19,30, 3,30,14
1170 DATA 4,16,17, 4,17,28, 4,28,20, 4,20,24, 4,24,16
1180 DATA 5,17,16, 5,16,25, 5,25,22, 5,22,29, 5,29,17
1190 DATA 6,19,18, 6,18,26, 6,26,21, 6,21,30, 6,30,19
1200 DATA 7,18,19, 7,19,31, 7,31,23, 7,23,27, 7,27,18
1210 DATA 8,20,21, 8,21,26, 8,26,12, 8,12,24, 8,24,20
1220 DATA 9,21,20, 9,20,28, 9,28,14, 9,14,30, 9,30,21
1230 DATA 10,23,22, 10,22,25, 10,25,13, 10,13,27, 10,27,23
1240 DATA 11,22,23, 11,23,31, 11,31,15, 11,15,29, 11,29,22

READY.
```

End Listing Seven

Listing Eight

```
0000                  ; KEYSORT - RELOCATABLE BUBBLE SORT USING KEY ARRAY
0000                  ; POINTING TO INTEGER ARRAY
0000                  ;
0000                  ;  RICHARD L. RYLANDER    1/12/85
0000                  ;
0000                  ORIGIN=$CF59           ; 53081. (FOLLOWING DOS 5.1)
0000                  ;
0000                  KB     = $FB           ; 251. POINTER TO KEY ARRAY
0000                  ZB     = $FD           ; 253. POINTER TO DATA ARRAY
0000                  MAX    = $8C           ; 140. POKE WITH MAX ARRAY INDEX
0000                  TOP    = $AC
0000                  TOPDIS = $AD
0000                  FLAG   = $AE
0000                  NXTFLG = $61
0000                  CRRNT  = $62
0000                  REPEAT = $64
0000                  ;
0000                         *=ORIGIN
CF59                  ;
CF59   A0 FF          INIT   LDY #$FF        ; INITIALIZE KEY ARRAY
CF5B   C8             INLOOP INY
CF5C   98                    TYA
CF5D   91 FB                 STA (KB),Y
CF5F   C5 8C                 CMP MAX
CF61   D0 F8                 BNE INLOOP
CF63                  ;
CF63   85 AD          SORT   STA TOPDIS      ; 'A' HOLDS 'MAX'
CF65   A5 AD          LOOP1  LDA TOPDIS
CF67   85 AC                 STA TOP
CF69   A2 00                 LDX #0
CF6B   86 61                 STX NXTFLG
CF6D   86 AE                 STX FLAG
CF6F   86 64          LOOP2  STX REPEAT
CF71                  ;
CF71                  ; GET BOTH BYTES OF INTEGER POINTED TO BY
CF71                  ; 'KEY' ELEMENT.  RETURN WITH MSB ON STACK
CF71                  ; AND LSB IN THE ACCUMULATOR
CF71                  ;
CF71   8A             GETINT TXA
CF72   A8                    TAY
CF73   B1 FB                 LDA (KB),Y
CF75   0A                    ASL A
CF76   90 04                 BCC LOAD
CF78   C6 61                 DEC NXTFLG
CF7A   E6 FE                 INC ZB+1
CF7C   A8             LOAD   TAY
CF7D   B1 FD                 LDA (ZB),Y
CF7F   48                    PHA
CF80   C8                    INY
CF81   B1 FD                 LDA (ZB),Y
CF83   24 61                 BIT NXTFLG
CF85   10 04                 BPL NODEC
CF87   E6 61                 INC NXTFLG
CF89   C6 FE                 DEC ZB+1
CF8B   E4 64          NODEC  CPX REPEAT
CF8D   D0 08                 BNE ORDER
CF8F   85 62                 STA CRRNT
CF91   68                    PLA
CF92   85 63                 STA CRRNT+1
CF94   E8                    INX
CF95   D0 DA                 BNE GETINT
CF97                  ;
CF97                  ; COMPARE INTEGERS OBTAINED THROUGH KEY ARRAY
CF97                  ; IF 'CURRENT' >= 'NEXT' THEN SWAP KEY
CF97                  ; ELEMENTS, ELSE CONTINUE
CF97                  ;
CF97   C5 62          ORDER  CMP CRRNT
CF99   68                    PLA
CF9A   E5 63                 SBC CRRNT+1
CF9C   50 02                 BVC TEST
CF9E   49 80                 EOR #$80
CFA0   10 13          TEST   BPL NOSWAP
CFA2   8A             SWAP   TXA
CFA3   A8                    TAY
CFA4   86 AD                 STX TOPDIS
CFA6   B1 FB                 LDA (KB),Y
CFA8   48                    PHA
CFA9   88                    DEY
CFAA   B1 FB                 LDA (KB),Y
CFAC   C8                    INY
CFAD   91 FB                 STA (KB),Y
CFAF   68                    PLA
CFB0   88                    DEY
CFB1   91 FB                 STA (KB),Y
```

(Continued on page 82)

May 1985, #103

Dr. Dobb's Journal

Reader Service Card

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP., MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify)____________

B. Which languages are you MOST interested in?

☐ BASIC	☐ C	☐ PL/1
☐ Fortran	☐ LISP	☐ APL
☐ COBOL	☐ Prolog	☐ Logo
☐ Pascal	☐ Ada	☐ Smalltalk
☐ Modula-2	☐ Forth	Other ______

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other

D. Please indicate which of the following microcomputers you currently own and/or plan to buy in the next 12 months.

	Own	Plan to Buy
Apple	☐	☐
Commodore	☐	☐
Digital Equipment/DEC	☐	☐
Heath/Zenith	☐	☐
Hewlett-Packard	☐	☐
IBM	☐	☐
Macintosh	☐	☐
Radio Shack/Tandy TRS 80	☐	☐
Texas Instruments	☐	☐
Other (Specify)	☐	☐
None	☐	☐

E. What best describes the work you do with this microcomputer?

- ☐ Business functions
- ☐ Software/Hardware development
- ☐ Scientific/Engineering/R&D applications

F. Are you the decision maker or very influential in computer-related purchases at work?

☐ Yes ☐ No

G. Is your company a dealer, distributor, or systems house for microcomputers?

☐ Yes ☐ No

H. Subscriber

☐ Yes ☐ No

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. One card per person.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135

Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

This offer expires August 31, 1985

Name ____________________ **Phone (____)**__________

Address ____________________

City/State/Zip ____________________

May 1985, #103

Dr. Dobb's Journal

Reader Service Card

A. Your primary job function: (Check one only)

- ☐ Company management (Pres., V.P., Treas., Owner, Gen. Mgr., Mktg. Dir.)
- ☐ Computer systems management (V.P. EDP., MIS Director, Data Processing Mgr., Data Communications Mgr., Network Planner)
- ☐ Engineering management (V.P. Engrg., Chief Engr., Tech. Director, Dir. R&D)
- ☐ Systems integrators (Systems Designer, Project Engr., Systems Application Engr., Technical Staff Members)
- ☐ Consultants (Computer/EDP/Data Communications Consultant)
- ☐ Educators (Educational users and instructors of computer technology)
- ☐ Systems/Programming specialists—mini-micro systems
- ☐ Other (Please specify)____________

B. Which languages are you MOST interested in?

☐ BASIC	☐ C	☐ PL/1
☐ Fortran	☐ LISP	☐ APL
☐ COBOL	☐ Prolog	☐ Logo
☐ Pascal	☐ Ada	☐ Smalltalk
☐ Modula-2	☐ Forth	Other ______

C. What is the operating system?

- ☐ CP/M (or derived)
- ☐ UNIX (or derived)
- ☐ MS-DOS (or derived)
- ☐ Other

D. Please indicate which of the following microcomputers you currently own and/or plan to buy in the next 12 months.

	Own	Plan to Buy
Apple	☐	☐
Commodore	☐	☐
Digital Equipment/DEC	☐	☐
Heath/Zenith	☐	☐
Hewlett-Packard	☐	☐
IBM	☐	☐
Macintosh	☐	☐
Radio Shack/Tandy TRS 80	☐	☐
Texas Instruments	☐	☐
Other (Specify)	☐	☐
None	☐	☐

E. What best describes the work you do with this microcomputer?

- ☐ Business functions
- ☐ Software/Hardware development
- ☐ Scientific/Engineering/R&D applications

F. Are you the decision maker or very influential in computer-related purchases at work?

☐ Yes ☐ No

G. Is your company a dealer, distributor, or systems house for microcomputers?

☐ Yes ☐ No

H. Subscriber

☐ Yes ☐ No

To obtain information about products or services mentioned in this issue, circle the appropriate number listed below. Use bottom row to vote for best article in issue. One card per person.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135

Articles: 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214

This offer expires August 31, 1985

Name ____________________ **Phone (____)**__________

Address ____________________

City/State/Zip ____________________

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

P.O. BOX 13851
PHILADELPHIA, PA 19101

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT #27346, PHILADELPHIA, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS
Dr. Dobb's Journal

P.O. BOX 13851
PHILADELPHIA, PA 19101

Drawing on the C-64 (Listing Continued, text begins on page 50)

Listing Eight

```
CFB3  E6 AE              INC FLAG
CFB5  E4 AC      NOSWAP  CPX TOP
CFB7  D0 B6              BNE LOOP2
CFB9  A5 AE              LDA FLAG
CFBB  D0 A8              BNE LOOP1
CFBD             ;
CFBD             ; UNPACK THE BYTE ELEMENTS OF THE 'KEY' ARRAY
CFBD             ; INTO BASIC'S NORMAL 2-BYTE INTEGER FORMAT
CFBD             ;
CFBD  A6 BC      UNPACK  LDX MAX
CFBF  E8                 INX
CFC0  CA         PKLOOP  DEX
CFC1  8A                 TXA
CFC2  A8                 TAY
CFC3  B1 FB              LDA (KB),Y
CFC5  48                 PHA
CFC6  8A                 TXA
CFC7  0A                 ASL A              ; MOVE TO 2*I+1
CFC8  09 01              ORA #1
CFCA  90 04              BCC STORE
CFCC  E6 61              INC NXTFLG
CFCE  E6 FC              INC KB+1
CFD0  A8         STORE   TAY
CFD1  68                 PLA
CFD2  91 FB              STA (KB),Y
CFD4  A9 00              LDA #0
CFD6  88                 DEY
CFD7  91 FB              STA (KB),Y
CFD9  A5 61              LDA NXTFLG
CFDB  F0 04              BEQ OK
CFDD  C6 61              DEC NXTFLG
CFDF  C6 FC              DEC KB+1
CFE1  8A         OK      TXA
CFE2  D0 DC              BNE PKLOOP
CFE4  60         DONE    RTS
CFE5             .END

ERRORS = 00000


SYMBOL TABLE

SYMBOL VALUE
 CRRNT    0062   DONE     CFE4   FLAG     00AE   GETINT   CF71
 INIT     CF59   INLOOP   CF5B   KB       00FB   LOAD     CF7C
 LOOP1    CF65   LOOP2    CF6F   MAX      008C   NODEC    CF8B
 NOSWAP   CFB5   NXTFLG   0061   OK       CFE1   ORDER    CF97
 ORIGIN   CF59   PKLOOP   CFC0   REPEAT   0064   SORT     CF63
 STORE    CFD0   SWAP     CFA2   TEST     CFA0   TOP      00AC
 TOPDIS   00AD   UNPACK   CFBD   ZB       00FD

END OF ASSEMBLY
```

End Listing Eight

Listing Nine

```
0000             ;   "WRITE"   RICHARD L. RYLANDER
0000             ;              12/30/84
0000             ; REVISED  1/19/85  - ORIGIN MOVED TO $CFE5 (53221.)
0000             ;
0000             ;  PUT TEXT CHARACTERS ON GRAPHIC SCREEN
0000             ;  (UNDER BASIC ROM) IN VARIOUS STYLES
0000             ;
0000             *=$CFE5                   ; PUT CODE AFTER DOS 5.1
CFE5  A5 01      WRITE  LDA $01            ; SWITCH OUT BASIC ROM
CFE7  29 FE             AND #$FE
CFE9  85 01             STA $01
CFEB  A0 07             LDY #7
CFED  B1 FD      LOOP   LDA ($FD),Y        ; READ CHARACTER BYTE
CFEF  31 FB             AND ($FB),Y        ; MODIFY W/SCREEN BYTE
CFF1  91 FB             STA ($FB),Y        ; STORE IN SCREEN
CFF3             ;
CFF3             ; POKE NEW LOGICAL OPERATOR TO REPLACE
CFF3             ; 'AND' (53231.) FOR DIFFERENT STYLES
CFF3             ; ORA=17.  BIT (NOP)=36.  AND=49.  EOR=81.
CFF3             ;
CFF3  88                DEY
CFF4  10 F7             BPL LOOP
CFF6  A5 01             LDA $01            ; RESTORE BASIC ROM
CFF8  09 01             ORA #1
CFFA  85 01             STA $01
CFFC  60                RTS
CFFD             .END

ERRORS = 00000

SYMBOL TABLE

SYMBOL VALUE
 LOOP     CFED   WRITE    CFE5

END OF ASSEMBLY
```

End Listings

JOIN US AT NCC '85

Today's Access Code to Tomorrow's Technology.

Featuring Keynote Speaker Admiral B.R. Inman. NCC '85, July 15-18, Chicago, Illinois.

The "Leading Edge" conference . . .

Don't miss the 1985 National Computer Conference, July 15-18 at McCormick Place and McCormick Place West, Chicago, Illinois, a key industry forum for new product and services introductions.

NCC '85 features a Keynote Address by Admiral B.R. Inman, President and CEO of Microelectronics and Computer Technology Corporation of Austin, Texas.

His topic, "Managing the Creation and Commercialization of Technology" will deal with the industry-wide challenge of converting technical advances into commercial successes.

A challenge everyone in the information technology field faces.

More than just a trade show . . .

At NCC '85 you get more than just exhibits. You can attend informative Technical Sessions on subjects such as artificial intelligence and software systems. You can also attend Professional Development Seminars, and special events like Pioneer Day and our Early Bird Reception. Plus, you can explore, experience and exchange ideas while gathering important information for future buying decisions.

Top names in the industry will be there—like IBM, Cullinet and McGraw Hill—showcasing their latest innovations and hottest new developments.

Whatever your special interest or area of expertise, you'll find the state-of-the-art in hardware, software and services at NCC '85!

Tomorrow's technology is just a phone call away . . .

To attend NCC '85 all you have to do is pick up the phone and call toll-free 800-NCC-1985. (Or fill out the coupon below.) And, if you register early, you get a $25 discount on the Full Conference Registration fee!

Don't miss this opportunity to be on the "Leading Edge." Plan on attending NCC '85!

CALL TOLL-FREE 800-NCC-1985
(Or fill out the coupon below.)

1985 National Computer Conference
Technology's Expanding Horizons
☐ **Yes!** I want to attend NCC '85. Please send me a registration form today!

Name
Title
Company
Address
City State Zip

Mail to: NCC '85 • 1899 Preston White Dr. • Reston, VA 22091 TB

OR CALL TOLL-FREE 800-NCC-1985 AND REGISTER TODAY! Visa, MasterCard, and American Express accepted. Register early and save $25!

A Compiler Written in Prolog

by G. A. Edgar

Why would anyone write a compiler in a language like Prolog? Actually, the compiler seems to have come out rather well. I wrote it primarily as a learning exercise for Micro-PROLOG.

Micro-PROLOG

I first heard of the Prolog programming language in connection with the Japanese effort to develop the "fifth generation" in computing. At the time, I read a few things about Prolog, such as the book by Clocksin and Mellish.[1] My impression was that Prolog is an interesting language (I was right), but that it probably is not practical to implement PROLOG on a microcomputer (was I ever wrong!).

D. E. Cortesi, in "Dr. Dobb's Clinic" for May 1984, gave a half-page description of Micro-PROLOG. I wrote for information on it and by July was running it on my CP/M Z80 computer. Micro-PROLOG is also available for MSDOS, PCDOS, CP/M-86, and Unix.

Micro-PROLOG comes with an IBM PC-size loose-leaf manual with more than 200 pages. This is a complete reference to the language but not a good tool for learning. Micro-PROLOG, however, also comes with a 400-page soft-cover tutorial, which is a good place for a beginner to start. Learning the language will take a substantial effort for a programmer experienced only in Pascal, C, or PL/I. The tutorial[2] is published separately as a book; if you are curious, it should be available in most large university libraries.

After I began to get a feel for the language, I used it to solve a few logic puzzles. For example, Lewis Carroll's logic book[3] includes some puzzles at the end as a teaser for a later book that was never published. They are much too complicated to solve simply by common sense reasoning, but I can now say definitively that "All of the Monitors are awake."

If Prolog were good only for logic problems, it would not be of wide interest. So I looked around for some other relatively easy task to use as a programming exercise. A compiler for VALGOL fit the need nicely. It improved my understanding of Micro-PROLOG and my understanding of what a compiler does.

The early returns on our March special issue on Prolog are encouraging—so much so that we're offering this excursion into programming in logic.

This compiler is described below. It translates VALGOL I, also described below, into 8080 assembly language. Of course, you would learn more by undertaking such a programming project yourself than you will by examining mine, but it serves as proof that you can use Micro-PROLOG for something unexpected.

Using Micro-PROLOG

Before I describe the compiler, let me give a brief description of how Micro-PROLOG differs from other versions of Prolog.

G. A. Edgar, 107 W. Dodridge St., Columbus, OH 43202

Upper- and lower-case letters are considered different, and the hyphen is treated like a letter. Variables are X, Y, Z, x, y, z, or one of these six letters followed by a positive integer, such as X132 or z22. Other strings of letters and numbers are identifiers for constants.

Micro-PROLOG consists primarily of a pattern matcher, plus a few other built-in routines, such as a routine for I/O. The way Micro-PROLOG looks depends on which of the front ends you are using. Each front-end program is written in Prolog. If you are using the SIMPLE front end, one-place predicates can be given either in prefix form

```
alphanumeric(X)
```

or in postfix form

```
X alphanumeric
```

Two-place predicates can be given either in prefix form

```
LESS(@ X)
```

or in infix form

```
@ LESS X
```

Predicates with three or more arguments must be in prefix form. Thus, a Prolog program entered via SIMPLE tends to look like this:

```
labelused (0)
X alphanumeric if X alphabetic
X alphanumeric if X digit
X alphabetic if @ LESS X and X
    LESS [
X alphabetic if ' LESS X and X
    LESS {
```

One useful object in Micro-PROLOG is the list. The notation for a list is to enclose the items between parentheses: (A B C 1 2 3). Some of the items might be lists themselves: (A (B C) (1 2 3)).

The 16-bit versions of Micro-PROLOG include a front-end supervisor that imitates the DEC-10 Prolog:

```
labelused(0).
alphanumeric(X) :- alphabetic (X).
alphanumeric(X) :- digit(X).
```

The form in which an operator is used is declared using the types fx, xf, xfy, and so on (see Clocksin and Mellish, page 91). DEC-10 lists are built using square brackets: [a,b,c].

The standard Micro-PROLOG syntax simply uses lists. In this regard, it is somewhat like LISP. The list (labelused 0), which indicates the predicate name "labelused" applied to the argument "0," is an example of an atom. A clause is a list of atoms. You add things to the data base in this way:

```
((labelused 0))
((alphanumeric X) (alphabetic X))
((alphanumeric X) (digit X))
```

This may seem confusing at first, but after a while it becomes more or less routine.

This minimal introduction to the Micro-PROLOG interpreter should be enough for you to gain some understanding of the VALGOL compil-

er, but you will need a much more complete knowledge of Micro-PROLOG to understand all of the details.

VALGOL I

The language VALGOL I (very small ALGOL?) is a derivative of ALGOL-60. D. V. Schorre described it in 1964 as a sample language for the compiler-writing language META II. His paper[4] was reprinted in *Dr. Dobb's Journal* (April 1980), which is where I found it. I had worked with VALGOL I in connection with the compiler-writing language Meta4 (in a SIG/M disk), so it seemed natural to use it for the same thing in Prolog.

VALGOL I has a few peculiarities. The keywords that usually are typeset in boldface here are preceded by a period (.begin, .end, .if, .then, .else, .until, .do, .integer). This also applies to the equals sign for comparing two expressions (.=). VALGOL I has only one data type, namely .integer, a 16-bit two's-complement number. The assignment statement is the reverse from the normal order (5 =: x assigns the value 5 to x). Arithmetic allows addition, subtraction, and multiplication, but there is no unary minus sign, so if you want – 2 you must write either 0 – 2 or 65534.

A more complete description of the syntax appears below in the description of the compiler.

Using the VALGOL I Compiler

The VALGOL I compiler is in a file called VALGOL.LOG. A typical use, to compile the short VALGOL program in the file P.VAL, is shown in Listing One (page 89). The command shown is in the standard Micro-PROLOG syntax. If the SIMPLE front end is also loaded, you could use the command:

is("P.VAL" compile "P.ASM")

The compiler prints out the input file P.VAL on the console as it works. The result of the compilation in the sample is P.ASM, included here as Listing Four (page 96).

If you are familiar with the Intel mnemonics for the 8080, you may find it instructive to compare the VALGOL input with the assembly language output to see how the compiler works.

The VALGOL Definition

Take a look at the compiler in Listing Two (page 90). I have repeated part of the compiler using the SIMPLE syntax in Listing Three (page 96). SIMPLE is unsuitable for the complete listing for several reasons. In a few places in the compiler, I have used "meta-variables," which are not supported by SIMPLE. A SIMPLE listing also would not keep the comments and the formatting, making it confusing to read. So try to understand instead the list notation used for the standard syntax of Micro-PROLOG in Listing Two.

The lists preceded by a ? are to be executed immediately when they are loaded; the others are to be stored in the data base. Because the predicate symbol (/ *) is a special one that is always true, a list in the file of the form

?((/ * anything goes here))

will serve as a comment. (This is a clumsy feature of Micro-PROLOG.)

The compiler listing is in two parts. The first part is independent of the particular syntax of VALGOL I and could be used equally well with many other programming languages. The second, below a horizontal line, is a VALGOL-specific part. In one sense, the part above the line implements an interpreter that runs the compiler described below the line. Let's examine a portion of the VALGOL-specific part of the code.

The message line, like all these lines, becomes a statement in the Prolog data base. The predicate symbol of the statement is "message," and the single argument is the string enclosed in quotation marks. The compiler will look for a message fact in the data base and print it out when a compilation starts.

The next line is the syntax line. This tells the compiler that the primary unit of syntax is the program.

The next statement is a description of a program. First comes the keyword .begin. When this is matched, seven opening lines are sent to the output file. (I have put the VALGOL syntax description on the left-hand side of the page and the ASM output description on the right-hand side. A VALGOL I compiler for another processor would keep the same left-hand side and change the right-hand side.) Then an opt-declaration (optional declaration, see below) is followed by one or more statements separated by semicolons. The multiple predicate allows matching of its arguments zero or more times. Finally comes the keyword .end. When this is matched, the final JMP is inserted in the output file, followed by the four subroutines and the stack space.

The opt-declaration could be a declaration followed by a semicolon, or (if that cannot be found) it could be empty, which matches the input file automatically. Inclusion of the "empty" alternative is a way to allow for no declaration. A declaration is the keyword .integer, followed by one or more identifiers separated by commas. For each identifier, the output file reserves a two-byte storage location.

A statement is one of the following:

(1) An I/O statement, either

edit(expression , 'string')[1]

which will send (expression) spaces and then the (string) to the console, or

print

which will send an end-of-line to the console.

(2) An assignment statement:

expression =: variable

(3) A loop:

```
.until expression
    .do statement
```

A value of zero for the expression is considered false, and a nonzero value is true.

(4) A conditional statement:

```
.if expression
    .then statement
    .else statement
```

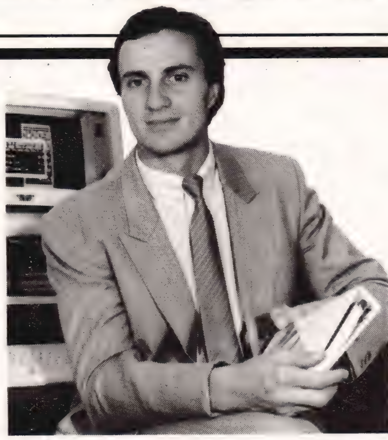
The .else is not optional.

(5) A block, which consists of the keyword .begin, followed by an optional declaration, followed by zero or more statements separated by semicolons, followed by the keyword

We're getting hardnosed at Softway.

From now on MATIS™ is only $49⁹⁵!

(MATIS, the complete User Interface development tool has been selling for $150.)



Denis Moran
President, Softway, Inc.

Why the radical price cut?

We decided after looking over the competition that MATIS had so many advantages it should be made available to more programmers. We decided to compete aggressively so you could easily afford to have MATIS in **your** bag of tricks. We hear from MATIS users in the USA and France that it is a truly loveable product. Sooo...we're running this big ad to promote our new low price.

MATIS windows are beautiful.

Display any portion of any screens you create at any point in your program. Scroll in any direction manually with cursor keys...or automatically.

And the screens are HUMUNGEOUS!

MATIS screens can be just about as big as you want...up to 65,534 rows by 65,534 columns! The number of screens is only limited by available memory.

Print big MATIS screens directly.

One command sends your screens to your printer with no need to program special routines when your virtual screen is bigger than your terminal screen.

User input fields are a snap.

Creating fields for data entry is easy with no limit to size or number by screen. Request for input separately or in blocks.

Control your keyboard with MATIS.

It keeps track of keys that are pressed during the execution of your program and lets you assign specific functions to selected keys.

Control that screen too!

MATIS is extremely versatile and flexible when it comes to controlling lines, columns, fields, and text. They can be modified, transferred, displayed or moved with a single command. All video attributes are supported: color, reverse video, blinking...you name it, you got it.

Want an interactive screen builder?

You've got it with MATIS. It's called "MATPAGE"™ and it lets you create and modify any of your screens in an interactive mode.

MATIS adds over 70 routines to your program.

Written in Assembler, MATIS routines are fast and powerful giving your program improved efficiency and enhanced visual appeal, while they reduce its size and maintenance worries. And MATIS separates screen design from the core of your program.

MATIS is unique.

We don't think there's a single program that combines as many tools in one package as completely or as well as MATIS. It interfaces with Intepreted and Compiled BASIC (Microsoft), C (Lattice, Microsoft, Aztec), PASCAL (IBM, Microsoft) and ASSEMBLER. All you need is an IBM* PC/XT or true compatible under DOS, 128k or RAM, monochrome or color monitor.

You get an easy to follow no-frills manual and a 30-Day Money Back Guarantee.

Late News: MATIS/T™ for TURBO-PASCAL** only $29.95

An indispensable add-on at a dynamite price. What more can we say?

Denis Moran

Denis Moran

™ MATIS, MATIS/T, & MATPAGE are Trademarks of Softway, Inc.
*IBM is a Reg. Trademark of IBM Corp.
**Turbo Pascal is a Reg. Trademark of Borland International

Softway, Inc.

500 Sutter Street, Suite 222 E
San Francisco, California 94102
(415) 397-4666

24-Hour Credit Card Orders By Phone:
1 (800) 227-2400 EXT 989 In California: 1(800) 772-2666 EXT 989

Please ship the following at once. I understand there is a 30-day money back guarantee.

_____ Copies of MATIS at $49.95 plus $3 shpg. $ _____
_____ Copies of MATIS/T for TURBO PASCAL at $29.95 plus $2 shpg. $ _____
California residents, add 6½% sales tax $ _____
☐ I like to read specs, so send me a folder. Total $ _____

Name _____
Address (please no P.O. Box) _____
City/State/ZIP _____
Phone (   ) _____ Signature _____
Make payment by money order, check, or charge card ☐ VISA ☐ MASTER CARD
Number _____ Exp. _____

Distributed in Europe By: MICRO APPLICATION SOFTWARE • 147 Avenue Paul Doumer, 92500 RUEIL MALMAISON FRANCE, Tel (1) 732.92.54

.end. The null statement

.begin .end

is allowed.

The following test for equality is allowed:

expression .= expression

This has value 1 if true and 0 if false. Expressions can be built up from variables, numbers, the operations +, −, *, and parentheses. Notice how the syntax near the end of Listing Two is arranged to take into account the precedence of these operations.

The Language-Independent Part

Now let's look at the first part of Listing Two (part of this is repeated in SIMPLE syntax as Listing Three). When Micro-PROLOG attempts to verify a compile atom, it looks first in the data base for a message fact; if it finds one, it verifies the assertion (P x), which is always true but which has the side effect of printing x on the console. The assertion PP is also always true and starts a new line at the console. The compile atom continues as it prints X –> Y, where X is the first argument (presumably the filename of the VALGOL input file) and Y is the second argument (the ASM output file).

Next the compiler deletes all facts about "labelused" (which may be left from a previous compile) and starts over with (labelused 0). It does something similar with "last-shown," which keeps track of how much of the file has been shown on the console.

Then the compiler opens the input file X and adds a fact to the data base so that later it can find the name of the input file. Similarly, it opens the output file Y. It skips white-space characters in the input file, starting at the beginning and ending at location Z1 (x in Listing Three). Then it finds the syntax fact to tell it what to start compiling with and feeds that to the Q routine, which carries out the actual compile.

Finally, the compiler closes the files and prints out a concluding message. If compilation fails for some reason, control goes to the second compile clause, and it prints out the error message.

Now look at Q. This short routine implements the recursive-descent compile, according to the code in the last half of the compiler. The first two arguments are the before and after file positions for both the input and output files. Whenever Q has to backtrack, the files automatically are rewound to the appropriate point for another try.

Notice the use of the variable Z (the third argument of Q) as the predicate symbol in a search of the data base. This is called a "meta-variable" in Micro-PROLOG: the name of the predicate to be verified is supplied as a variable, not as a constant. This powerful feature of Micro-PROLOG may not be available in other Prologs. The X1 after the vertical line, which indicates the rest of the list, is another meta-variable. The remainder of this half of the code comprises various subroutines for use in the compile.

This is a very brief description of only a small part of this compiler. A complete understanding requires a greater familiarity with the many features and quirks of the Micro-PROLOG package.

Conclusion

The compiler shown in Listing Two is written in the standard syntax of Micro-PROLOG. As such, it should run under any version of Micro-PROLOG. I would be interested in hearing from readers who get it to work under other versions. (Of course, the output would still be in 8080 assembly language.)

This compiler runs rather slowly. The common belief is that interpreters are slow, and this is an interpreter within an interpreter. The 10-line sample compilation in Listing One takes about 30 sec on my 4 MHz Z80. Because I wrote the compiler primarily as a programming exercise, I have not been overly concerned with a remedy for the speed problem. Probably the bottleneck occurs in the input routines that process one character at a time. For a faster compiler, routines such as match, id, or number would be recoded in assembly language. The Micro-PROLOG manual has instructions on how to link such new built-in functions in assembly code with Micro-PROLOG.

References

1. W. F. Clocksin and C. S. Mellish, *Programming in Prolog,* Springer-Verlag, 1981.
2. K. L. Clark and F. G. McCabe, *Micro-PROLOG: Programming in Logic,* Prentice-Hall, 1984.
3. Lewis Carroll, *Symbolic Logic,* 4th edition, 1896 (Reprinted by Dover Publications, 1958).
4. D. V. Schorre, "META II, a syntax-oriented compiler writing language," *Proc. 19th Nat. Conf. ACM,* 1964 (Reprinted in *Dr. Dobb's Journal,* April 1980, pp. 17–21; listings pp. 22–25).

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 195**.

Compiler in Prolog (Text begins on page 84)

Listing One

Use of the compiler

```
B>PROLOG LOAD VALGOL

micro-PROLOG 3.1 for RML 380/480Z 29/03/84
(c) 1984 Logic Programming Associates Ltd.
38517 Bytes Free

&&.?((compile "P.VAL" "P.ASM"))
VALGOL I compiler - translates VALGOL to ASM
P.VAL -> P.ASM

.begin
.integer x;
0 =: x;
.until x .= 15 .do
      .begin
      edit (1+14*x-x*x, '+');
      print;
      x+1 =: x
      .end
.end

** Compilation of "P.VAL" complete **
&.QT.

B>ASM P.BBZ

CP/M ASSEMBLER - VER 2.0
01F7
001H USE FACTOR
END OF ASSEMBLY

B>LOAD P


FIRST ADDRESS 0100
LAST  ADDRESS 01F6
BYTES READ    00B9
RECORDS WRITTEN 02

B>P

 +
              +
                         +
                                  +
                                         +
                                              +
                                                 +
                                                  +
                                                 +
                                              +
                                         +
                                  +
                         +
              +
 +

type p.val

.begin
.integer x ;
0 =: x ;
.until x .= 9 .do
        .begin
        edit ( x*x + 1 , '+' ) ;
        print ;
        x + 1 =: x
        .end
.end

A>p

 +
  +
     +
          +
                 +
                          +
                                     +
                                                  +
                                                                 +
```

End Listing One

(Listing Two begins on next page)

Listing Two

VALGOL.LOG

```
?(( /*
   VALGOL.LOG
   Compiler for VALGOL I -- ver. 1.2

   Written in Micro-PROLOG.
   (Tested with Micro-PROLOG version 3.1 for CP/M-80)
   Translates VALGOL I to ASM-compatible 8080 assembly language.
   Uses file SEEKs for backtracking.

        written by: G. A. EDGAR    status: public domain

   0.3     September 9, 1984      M80, Z80
   1.0     September 16, 1984     output for ASM
   1.1     November 1, 1984       display input
   1.2     November 10, 1984      version for DDJ

   Usage - in SIMPLE syntax:
        &.is("FOO.VAL" compile "FOO.ASM")
         - in standard syntax:
        &.?((compile "FOO.VAL" "FOO.ASM"))
))


?(( /* ---------- Language independent part ---------------))

?(( /* compile: input X, output Y))
((compile X Y)
    (message x) (P x) (PP) (P X "->" Y) (PP) (PP)
    (KILL labelused) (ADDCL ((labelused 0)) )
    (KILL last-shown) (ADDCL ((last-shown (-1|-1))) )
    (OPEN X) (KILL infile) (ADDCL ((infile X)) )
    (erase Y) (CREATE Y) (KILL outfile) (ADDCL ((outfile Y)) )
    (skip ((0|0)|(0|0)) Z1 ) (syntax z) (Q Z1 Z z)
    (CLOSE X) (CLOSE Y)
    (PP) (PP ** Compilation of X complete **))
((compile X Y)
    (PP) (PP ** Syntax error **)
    (CLOSE X) (CLOSE Y))
((C X Y)
    (compile X Y))

?(( /* Q:  find it in the language-specific database))
((Q X Y Z)
    (Z | X1) (sequential X Y | X1))
((sequential X X))
((sequential X Y (z|Z)|y)
    (z X X1|Z) / (sequential X1 Y|y))

?(( /* out: send a line to outfile))
((out (y1|z1) (y1|z2) | X)
    (outfile Z) (SEEK Z z1) (outx Z | X)
    (outx Z "^M^J") (SEEK Z z2))
((outx Z))
((outx Z X | x)
    (W Z (X)) / (outx Z | x))

?(( /* match: the input matches string Z))
((match X Y Z)

    (STRINGOF Z1 Z) / (matchx X X2 Z1) (skip X2 Y))
((matchx X Y (x|Z))
    (inchar X X1 x1) (EQ x x1) / (matchx X1 Y Z))
((matchx X X ()))

?(( /* empty: matches automatically))
((empty X X))

?(( /* multiple: match the following zero or more times))
((multiple X Y|Z)
    (once X X1|Z) / (multiple X1 Y|Z))
((multiple X X|Z))
((once X X))
((once X Y (z|Z)|y)
    (z X X1|Z) / (once X1 Y|y))

?(( /* label: generate a new label))
((label X X Y)
    (labelused Y1) (SUM Y1 1 Y) (KILL labelused)
    (ADDCL ((labelused Y)) ))
```

(Continued on page 92)

Listing Two

```
?(( /* string: match a string quoted within characters x))
((string X Y x Z)
    / (inchar X X2 x1) (EQ x x1) /
    (stringx X2 X3 x Z1) (skip X3 Y) (STRINGOF Z1 Z))
((stringx X Y x ())
    (inchar X Y x1) (EQ x x1) / )
((stringx X Y x (y | Z))
    (inchar X X1 y) / (stringx X1 Y x Z))

?(( /* id: match an identifier))
((id X Y Z)
    (idx X X2 Z1) (skip X2 Y) (STRINGOF Z1 Z))
((idx X Y (x|Z))
    (inchar X X1 x) (alphabetic x) / (alphanum X1 Y Z))
((alphanum X Y (x|Z))
    (inchar X X1 x) (alphanumeric x) / (alphanum X1 Y Z))
((alphanum X X ()))

?(( /* number: match a number))
((number X Y Z)
    / (numberx X X2 Z1)
    (NOT EQ Z1 ()) (skip X2 Y) (STRINGOF Z1 Z))
((numberx X Y (x|Z))
    (inchar X X1 x) (digit x) / (numberx X1 Y Z))
((numberx X X ()))

((alphanumeric x)
    (alphabetic x))
((alphanumeric x)
    (digit x))

((alphabetic x)
    (LESS "@" x)(LESS x "["))
((alphabetic x)
    (LESS "`" x)(LESS x "{"))

((digit x)
    (LESS "/" x)(LESS x ":"))

?(( /* skip: skip over characters listed as "skipchar"))
((skip X Y)
   (inchar X X1 x) (skipchar x) / (skip X1 Y))
((skip X X))

((skipchar " "))
((skipchar "^I"))
((skipchar "^J"))
((skipchar "^M"))

?(( /* inchar: input one character, show on console first time))
((inchar (y1|z1) (y2|z1) x)
   (infile Y) (SEEK Y y1) (RDCH Y x) (SEEK Y y2) (last-shown y3)
   (IF (before y3 y2)
     ((P x) (KILL last-shown) (ADDCL ((last-shown y2)) ))
     ()))

?(( /* before: compare two file positions ))
((before (y1|z1) (y1|z2))
   / (LESS z1 z2))
((before (y1|z1) (y2|z2))
   (LESS y1 y2))

?(( /* erase: erase a file, no failure))
((erase Y)
   (ERA Y) /)
((erase Y))

?(( /* --- VALGOL specifics ------------------------------ ))

((message "VALGOL 1 compiler - translates VALGOL to ASM"))
((syntax PROGRAM))

((PROGRAM
   (match ".begin")            (out    "VCPM   EQU     0")
                               (out    "VBDOS  EQU     5")
                               (out    "VTPA   EQU     256")
                               (out    "VCR    EQU     13")
                               (out    "VLF    EQU     10")
                               (out    "       ORG     VTPA")
                               (out    "       LXI     SP,VSTACK")
   (Q OPT-DECLARATION)
   (Q STATEMENT)
```

```
(multiple
    (match ";")
    (Q STATEMENT))
(match ".end")
                (out    "        JMP     VCPM")
                (out    "VMULT:")
                (out    "        MOV     B,H")
                (out    "        MOV     C,L")
                (out    "        XRA     A")
                (out    "        MOV     H,A")
                (out    "        MOV     L,A")
                (out    "        MVI     A,16")
                (out    "VMULT1: PUSH    PSW")
                (out    "        DAD     H")
                (out    "        XRA     A")
                (out    "        MOV     A,C")
                (out    "        RAL")
                (out    "        MOV     C,A")
                (out    "        MOV     A,B")
                (out    "        RAL")
                (out    "        MOV     B,A")
                (out    "        JNC     VMULT2")
                (out    "        DAD     D")
                (out    "VMULT2: POP     PSW")
                (out    "        DCR     A")
                (out    "        ORA     A")
                (out    "        JNZ     VMULT1")
                (out    "        RET")
                (out    "VEDIT:")
                (out    "        MOV     A,H")
                (out    "        ORA     L")
                (out    "        JZ      VEDIT1")
                (out    "        MVI     A,' '")
                (out    "        CALL    VCPMOUT")
                (out    "        DCX     H")
                (out    "        JMP     VEDIT")
                (out    "VEDIT1: POP     H")
                (out    "VEDIT2: MOV     A,M")
                (out    "        CPI     0")
                (out    "        INX     H")
                (out    "        JZ      VEDIT3")
                (out    "        CALL    VCPMOUT")
```

(Continued on next page)

Listing Two

```
                                      (out     "        JMP      VEDIT2")
                                      (out  "VEDIT3:  PUSH     H")
                                      (out     "        RET")
                                      (out    "VPRINT:")
                                      (out     "        MVI      A,VCR")
                                      (out     "        CALL     VCPMOUT")
                                      (out     "        MVI      A,VLF")
                                      (out     "        CALL     VCPMOUT")
                                      (out     "        RET")
                                      (out    "VCPMOUT:")
                                      (out     "        PUSH     H")
                                      (out     "        MOV      E,A")
                                      (out     "        MVI      C,2")
                                      (out     "        CALL     VBDOS")
                                      (out     "        POP      H")
                                      (out     "        RET")
                                      (out     "        DS       60")
                                      (out  "VSTACK:  DW       0")
                                      (out     "        END") ))

((OPT-DECLARATION
    (Q DECLARATION)
    (match ";")))
((OPT-DECLARATION
    (empty)))

((DECLARATION
    (match ".integer")                (label X1)
                                      (out     "        JMP      V" X1)
    (Q ID-SEQUENCE)                   (out    "V" X1 ":") ))

((ID-SEQUENCE
    (Q IDENTIFIER)
    (multiple
        (match ",")
        (Q IDENTIFIER) )))

((IDENTIFIER
    (id Z)                            (out   Z "V:    DS       2") ))

((STATEMENT
    (Q IO-STATEMENT)))
((STATEMENT
    (Q ASSIGNMENT-STATEMENT)))
((STATEMENT
    (Q UNTIL-STATEMENT)))
((STATEMENT
    (Q CONDITIONAL-STATEMENT)))
((STATEMENT
    (Q BLOCK)))

((BLOCK
    (match ".begin")
    (Q DECL-OR-ST)
    (multiple
        (match ";")
        (Q STATEMENT) )
    (match ".end") ))
((BLOCK
    (match ".begin")
    (match ".end") ))

((DECL-OR-ST
    (Q DECLARATION)))
((DECL-OR-ST
    (Q STATEMENT)))

((IO-STATEMENT
    (match "edit")
    (match "(")
    (Q EXPRESSION)
    (match ",")
    (string "'" Z)                    (out     "        CALL     VEDIT")
                                      (out     "        DB       '" Z "',0")
    (match ")") ))
((IO-STATEMENT
    (match "print")                   (out     "        CALL     VPRINT") ))

((CONDITIONAL-STATEMENT
    (match ".if")                     (label X1) (label X2)
    (Q EXPRESSION)
    (match ".then")                   (out     "        MOV      A,H")
```

```
                                        (out    "        ORA      L")
                                        (out    "        JZ       V" X1)
   (Q STATEMENT)
   (match ".else")                      (out    "        JMP      V" X2)
                                        (out    "V" X1 ":")
   (Q STATEMENT)                        (out    "V" X2 ":") ))

((UNTIL-STATEMENT
   (match ".until")                     (label X1) (label X2)
                                        (out    "V" X1 ":")
   (Q EXPRESSION)
   (match ".do")                        (out    "        MOV      A,H")
                                        (out    "        ORA      L")
                                        (out    "        JNZ      V" X2)
   (Q STATEMENT)                        (out    "        JMP      V" X1)
                                        (out    "V" X2 ":") ))

((ASSIGNMENT-STATEMENT
   (Q EXPRESSION)
   (match "=:")
   (id Z)                               (out    "        SHLD     " Z "V") ))

((EXPRESSION
   (Q EXPRESSION1)
   (Q OPT-RIGHT-SIDE)))

((OPT-RIGHT-SIDE
   (match ".=")                         (label X1) (label X2)
                                        (out    "        PUSH     H")
   (Q EXPRESSION1)                      (out    "        POP      D")
                                        (out    "        MOV      A,L")
                                        (out    "        SUB      E")
                                        (out    "        JNZ      V" X2)
                                        (out    "        MOV      A,H")
                                        (out    "        SBB      D")
                                        (out    "        JNZ      V" X2)
                                        (out    "        LXI      H,1")
                                        (out    "        JMP      V" X1)
                                        (out    "V" X2 ":")
                                        (out    "        LXI      H,0")
                                        (out    "V" X1 ":") ))
((OPT-RIGHT-SIDE
   (empty)))

((EXPRESSION1
   (Q TERM)
   (multiple
      (Q SIGNED-TERM))))

((SIGNED-TERM
   (match "+")                          (out    "        PUSH     H")
   (Q TERM)                             (out    "        POP      D")
                                        (out    "        DAD      D")))
((SIGNED-TERM
   (match "-")                          (out    "        PUSH     H")
   (Q TERM)                             (out    "        POP      D")
                                        (out    "        MOV      A,E")
                                        (out    "        SUB      L")
                                        (out    "        MOV      L,A")
                                        (out    "        MOV      A,D")
                                        (out    "        SBB      H")
                                        (out    "        MOV      H,A")  ))

((TERM
   (Q PRIMARY)
   (multiple
      (match "*")                       (out    "        PUSH     H")
      (Q PRIMARY)                       (out    "        POP      D")
                                        (out    "        CALL     VMULT") )))

((PRIMARY
   (id Z)                               (out    "        LHLD     " Z "V")))
((PRIMARY
   (number Z)                           (out    "        LXI      H," Z)))

((PRIMARY
   (match "(")
   (Q EXPRESSION)
   (match ")") ))
```

End Listing Two

(Listing Three begins on next page)

Compiler in Prolog

Listing Three

(Listing Continued, text begins on page 84)

Part of VALGOL.LOG in SIMPLE syntax

```
X compile Y if
     message (Z) and
     P (Z) and
     PP and
     PP and
     KILL (labelused) and
     add ((0 labelused)) and
     KILL (last-shown) and
     add (((-1|-1) last-shown)) and
     OPEN (X) and
     KILL (infile) and
     add ((X infile)) and
     erase (Y) and
     CREATE (Y) and
     KILL (outfile) and
     add ((Y outfile)) and
     skip (((0|0) 0|0) x) and
     syntax (y) and
     QQ (Q x z y) and
     CLOSE (X) and
     CLOSE (Y) and
     PP and
     PP (*** Compilation of X complete **)
X compile Y if
     PP and
     PP (** Syntax error **) and
     CLOSE (X) and
     CLOSE (Y)
X alphanumeric if
     X alphabetic
X alphanumeric if
     X digit
X alphabetic if
     @ LESS X and
     X LESS [
X alphabetic if
     ` LESS X and
     X LESS {
X digit if
     / LESS X and
     X LESS :
```

End Listing Three

Listing Four

P.ASM

```
VCPM    EQU     0
VBDOS   EQU     5
VTPA    EQU     256
VCR     EQU     13
VLF     EQU     10
        ORG     VTPA
        LXI     SP,VSTACK
        JMP     V1
xV:     DS      2
V1:
        LXI     H,0
        SHLD    xV
V2:
        LHLD    xV
        PUSH    H
        LXI     H,15
        POP     D
        MOV     A,L
        SUB     E
        JNZ     V5
        MOV     A,H
        SBB     D
        JNZ     V5
        LXI     H,1
        JMP     V4
V5:
        LXI     H,0
V4:
        MOV     A,H
        ORA     L
        JNZ     V3
        LXI     H,1
        PUSH    H
        LXI     H,14
        PUSH    H
        LHLD    xV
        POP     D
        CALL    VMULT
        POP     D
        DAD     D
        PUSH    H
        LHLD    xV
        PUSH    H
        LHLD    xV
        POP     D
        CALL    VMULT
        POP     D
        MOV     A,E
        SUB     L
        MOV     L,A
        MOV     A,D
        SBB     H
        MOV     H,A
        CALL    VEDIT
        DB      '+',0
        CALL    VPRINT
        LHLD    xV
        PUSH    H
        LXI     H,1
        POP     D
        DAD     D
        SHLD    xV
        JMP     V2
V3:
        JMP     VCPM
VMULT:
        MOV     B,H
        MOV     C,L
        XRA     A
        MOV     H,A
        MOV     L,A
        MVI     A,16
VMULT1: PUSH    PSW
        DAD     H
        XRA     A
        MOV     A,C
        RAL
        MOV     C,A
        MOV     A,B
        RAL
        MOV     B,A
        JNC     VMULT2
        DAD     D
VMULT2: POP     PSW
        DCR     A
        ORA     A
        JNZ     VMULT1
        RET
VEDIT:
        MOV     A,H
        ORA     L
        JZ      VEDIT1
        MVI     A,' '
        CALL    VCPMOUT
        DCX     H
        JMP     VEDIT
VEDIT1: POP     H
VEDIT2: MOV     A,M
        CPI     0
        INX     H
        JZ      VEDIT3
        CALL    VCPMOUT
        JMP     VEDIT2
VEDIT3: PUSH    H
        RET
VPRINT:
        MVI     A,VCR
        CALL    VCPMOUT
        MVI     A,VLF
        CALL    VCPMOUT
        RET
VCPMOUT:
        PUSH    H
        MOV     E,A
        MVI     C,2
        CALL    VBDOS
        POP     H
        RET
        DS      60
VSTACK: DW      0
        END
```

End Listings

DDJ Classifieds

Software

PCBTAM

Communications Access Method

Allows an IBM PC or compatible to perform BISYNC communication. PCBTAM is a general purpose interrupt driven access method usable by any Microsoft language. Requires IBM BSCA card and PC-DOS.
- Source or object license
- Asynchronous version available (PCATAM)
- Modifiable for other USARTS.

For details write:

SYMBIOTIC

Symbiotic Protocol Converters, Inc.
1011 Clifton Avenue, Clifton, New Jersey 07013
(201) 777-8454

Olive Branch Software

COPY PROTECTION

SLK/F places an assembled or compiled program on a diskette with 4 different copy-resistant features in such a way that it runs normally, but cannot be copied by backup programs such as COPYPC. The rest of the diskette is available as normal, and DOS may be added. Price $150.
Olive Branch Software
1715 Olive Street
Santa Barbara, CA 93101
(805)569-1682

DriveLiner

Drive Alignment Test Program for CP/M 2.2/3.1 With Dysan 8″ SSSD Diagnostic Disk
$65 check/MO postpaid
Chandler Software, 273 W Shore Dr, Marblehead MA 01945 (617)631-4685

BRINGING YOUR PROGRAMS OUT OF THE CLOSET?

DISKETTES & DUPLICATION

Large or Small - Let's Talk!!

5¼″ DISKETTES
SS/DD 99¢ EA. DS/DD $1.10 EA.
Minimum order 250 disks.
8″ - Color Jackets - 3½″ CALL!!

Hundreds of professionals can't be wrong!!!

CALL TODAY!!
800-437-0500
800-435-7400 In CA
(408) 262-3008

SUPPLY LINE

780 Trimble Rd. • Suite 608
San Jose, CA 95131

TECMAR GRAPHICS LIBRARY

TEKMAR lets you do high-res graphics on your TECMAR Graphics Master. Features windowing, viewporting, clipping, axis rotation. Similar to Tektronix graphics. Includes screen dump/restore, Epson screen print, support for HP and Western Graphtec plotters. Includes three curve-fitting programs and graphics application SOURCE CODE. Requires MS-FORT 3.20, or Lahey F77L. Price: $195.

ADVANCED SYSTEMS CONSULTANTS
18653 Ventura Boulevard, Suite 351
Tarzana, California 91356
(818) 990-4942

Offering low cost advertising reaching thousands of computer programmers and professionals each month. Departments to choose from include:
Software
Hardware
Accessories/Supplies
Career Opportunities
Services
User Groups. Special categories are also available.

Dr. Dobb's Journal
is pleased to announce the
DDJ Classifieds

RATES: DISPLAY ADVERTISERS: Price per column inch $100. Ad must run in 3 consecutive issues.
LINE ADVERTISERS: Price per line $12 (35 characters per line including spaces). Minimum of 5 lines. 3 consecutive issues. Add $3 per line for boldface type. Add $25 if a background screen is desired.

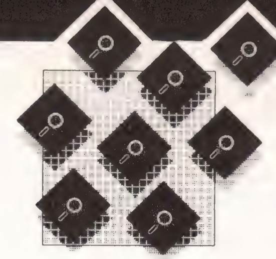
DISCOUNTS: Frequency discounts for 6 consecutive ads (less 7%) and for 12 consecutive ads (less 15%).

MECHANICAL REQUIREMENTS: Camera ready art or typewritten copy only (phone orders excepted). Display: Specify desired size, include $20 if reduction is required. Line: Specify characters in boldface, caps. Allow 6 weeks for publication.

PAYMENTS: Full payment in advance. Check, money order, Visa, M/C, AmEx.

Run this ad in ______ issues
Size: ______ col x ______ inches or 1 col x ______ lines
Payment by: ______ check ______ m/o ______ Visa ______ M/C ______ AmEx
Card # ______________________ Exp Date ______
Signature ______________________
Print Name ______________________
Company ______________________
Address ______________________
City ______________ St ______ Zip ______
Phone ______________ Current Subscriber ______

Mail to or phone Alex Williams, DDJ Classifieds, 2464 Embarcadero Way, Palo Alto, CA 94303 (415) 424-0600



Toolworks C/80 and Toolworks C/80 Mathpack, Version 3.1

Company: Software Toolworks, 15233 Ventura Blvd., Suite 1118, Sherman Oaks, CA 91403
Computer: Heath; Kaypro II, 4, and 10; Osborne 1 and the Executive; DEC VT-180 and Rainbow; Xerox 820
Price: Compiler, $49.95; Mathpack, $29.95
Circle Reader Service No. 101
Reviewed by D. C. Shoemaker

Few readers of *Dr. Dobb's Journal* are unaware of the contribution made to small computer system programmers by Ron Cain and his minimal C subset compiler. Over the years since its introduction, there have been many extensions and improvements, all building on his basic idea. One of the best and most complete versions is from Walt Bilofsky's Software Toolworks. First released in 1981, Version 3.1 is the most current and comes the closest to being a complete C compiler.

This software is a bargain at the price. The package comes in two parts, sold separately: the C compiler itself is $49.95, and the C/80 Mathpack is $29.95. There's a reason for the way the software is packaged and priced. Those just learning C probably have no need for the FLOAT and LONG data types that the Mathpack provides. Adding these features somewhat increases the size of the libraries and the assembled code. If you need them, Mathpack is a bargain-level entry into other more powerful capabilities given by the additional data types. Therefore, you don't have to pay for something you don't need, but you aren't limited to an entry-level compiler that can't do serious work.

C/80 is a complete implementation of the C language described by Brian Kernighan and Dennis Ritchie in *The C Programming Language* (Prentice-Hall, 1978) with the exception of the following features:

- FLOAT and LONG data types (available in the C/80 Mathpack)
- DOUBLE data type
- typedef
- Arguments to #define macros
- #line
- Declarations within nested blocks
- Bit fields

Whether or not these omissions are significant depends a great deal on what you want to do with the software. If you want to learn the language and get a whiff of programming in C, the C/80 package is sufficient. If you want to write some fast-running utility software and don't want to invest the time in assembly language, these omissions will cause no problems.

C/80 supports structures, statics, initialization, casts, compile time evaluation of constant expressions, and all the other C language features. It's worth mentioning some of the features that C/80 does provide, however, because they're often lacking in other subsets:

- Unix-style I/O redirection
- Conventional C I/O and string library
- Formatted and random access file I/O
- Dynamic storage allocation
- Runtime execution profile capability
- In-line assembly language capability

In addition, C/80 can generate source code compatible with Macro-80 and RMAC. Should you have neither, a serviceable assembler comes with the C/80 package. It works well and does the job with no fuss.

If you have a version of C/80 earlier than 3.0, here are the main changes in version 3.1:

- An expanded runtime library
- ROMable code
- RMAC compatibility
- A menu-configurable compiler
- \ at the end of a line for continuation
- #ifneed for selective compilation
- True alloc/free
- Command line wildcard expansion
- CP/M files now written in 128-byte records
- File 0 always the terminal

The changes from version 3.0 to 3.1 are to correct minor bugs and speed code generation. Some new compiler switches have been added, and you are now prohibited from reading from and writing to the same open file. If you have one of the earlier versions of C/80, return the original disk to Software Toolworks with $10.00, and they will give you the most recent version. I bought my first copy of C/80 in 1981 and have updated it twice.

Software Toolworks makes the C/80 package available in a variety of formats. Walt Bilofsky was one of the first to support Heath's in-house disk operating system HDOS, and he still does. However, the standard remains CP/M, and Software Toolworks provides disks for Heath, both hard and soft sector; Kaypro II, 4, and 10; Osborne I and the Osborne Executive; DEC's VT-180 and the Rainbow; the Xerox 820; and the standard 8-inch CP/M single-density format. There's

even a version for the Epson QX-10. Currently no version is offered for the IBM PC, PCjr, or any MSDOS, PCDOS, or ZDOS computer because the programs are written in 8080 code, but I expect that will change. Incidentally, I'm using the CP/M version on a Heath H120 running CP/M-85 with no trouble at all. Just order the soft-sector Heath version.

Using C/80 is simple. The hard part (as usual) is writing the original C language program. Then you call C/80 and let it compile the source into 8080 assembly language code. Let's use one of the test files on the distribution disk, HELLO.C. We do this by typing "C80 HELLO." The output file from C/80 can be modified if you wish. I'm not the greatest assembly language programmer, so I don't change the code, but someone more experienced might be able to tighten it up in some areas.

Next, call the assembler of your choice. I use the one that comes with the package. It's invoked as you might expect, by typing "AS HELLO." When the assembly is finished, you have an executable file called HELLO.COM. It will be rather large, due to the inclusion of pieces from the I/O and other libraries, but that's part of the tradeoff. The completed code executes relatively quickly. I don't have the capability to do much comparative benchmarking, but comparisons I've seen suggest that in some cases C/80 is a bit faster than BDS C and in others a bit slower. On balance they seem about equal, except that BDS C has some nonstandard features that get in the way of portability, and it costs twice what C/80 costs. BDS C doesn't provide as many formats, either.

Software Toolworks doesn't charge a license fee for the COM files created by C/80. This means that you're free to market whatever you write and compile as you see fit. Unlike some of the so-called "big boys" of the software industry, Bilofsky encourages programmers to market software using his tools.

A word about documentation: If you're a first-time C user, the manual that accompanies C/80 won't be too helpful. There are some examples in the documentation that you can play with, and several source code files on the distribution disks, but that's it for introductory material. Several good books on the market can help you get a better grasp of the language. The documentation lists eight of the most common of these, and new books appear almost monthly.

The C/80 manual is 50 pages long with a three-page index. It has 16 chapters, with most of the attention paid to a language summary, a description of the I/O and runtime libraries, and a discussion on running the compiler. The manual includes a brief discussion of the internal features of the compiler and useful tricks to bend the compiler to your will. There is also a detailed list of compiler error messages, what they mean, and what to do about them.

The C/80 Mathpack requires C/80 version 3.0 or greater. If you've been working with an earlier verson of C/80, the Mathpack is a good reason to take advantage of the Software Toolworks' update policy. Mathpack comes on one disk with a short manual. The Mathpack adds true 32-bit LONG and FLOAT data types to the C/80 compiler and runtime library. It includes a program to patch the C/80 compiler to recognize LONG and FLOAT, a runtime library to perform 32-bit arithmetic, routines to convert between ASCII and floating-point representation, an augmented printf, and a transcendental library written in C.

In practical terms, this gives the C/80 a 32-bit signed number capability in the range –2,147,483,648 to 2,147, 483, 647. Floating-point numbers have 24-bit precision, which equates to about seven decimal digits, and an exponent range of 10 to the plus or minus 38th power.

Installing the Mathpack requires only that you run the modifying program CCONFIGF.COM. A menu will appear with a range of options. Choose option N and press RETURN. This changes FLOAT and LONG from not available to available. Now type a Y in response to the question about making changes permanent, and you're done. Everything else about C/80 remains as before, except

that the size of the compiled and assembled code increases a bit.

The Mathpack manual is only 10 pages long with six chapters; it spends most of its time on installation and additions to the function library. You'll have to know how to make use of FLOAT and LONG before the Mathpack will do you much good.

In summary, if you want to learn C or if you already know a bit about the language and want an inexpensive but powerful implementation for your CP/M computer, my first choice would be the Software Toolworks C/80 and an introductory book. If you want an enhanced 32-bit capable C, add the Mathpack. For under $80.00 you can't miss.

Disk Maker I

Company: New Generation Systems, 1800 Michael Faraday Drive, Suite 206, Reston, VA 22090
(703) 471-5598,
(800) 368-3359
Operating System: CP/M 2.2
Price: $1695.00
Circle Reader Service No. 103
Reviewed by Jim Kronman

Disk Maker I from New Generation Systems offers a practical solution to the virtual Tower of Babel that exists in the world of CP/M and MSDOS disk formats. Acting as a peripheral device in an S-100 system, Disk Maker I can format, read, write, and duplicate over 170 disk formats (at the time of this review—the total grows every month or two), allowing you to copy virtually any soft-sectored format and transfer disk files between various CP/M and MSDOS disk formats. An optional file transfer utility program transfers files from DecMate II and Wang OIS word processing disk formats.

The standard Disk Maker I package consists of an S-100 disk controller board capable of supporting up to four disk drives, a 48 track-per-inch (tpi), double-sided 5¼-inch disk drive with power supply in a fan-cooled cabinet, and the Disk Maker software. Other disk drives for Disk Maker are a 96 tpi 5¼-inch drive, a 135 tpi 3½-inch drive, or a 96 tpi 1.2 Mb IBM PC/AT drive (as used in the IBM PC/AT) installed in the cabinet with the 48 tpi drive. A double-sided double-density (DSDD) 8-inch drive with its power supply in a separate cabinet is also available. Software options are the word processing transfer utilities and disk drive test software.

If you don't have an S-100 system, do not stop reading: New Generation Systems also offers a stand-alone Disk Maker II system. It is a 6 MHz, Z80, 64K CP/M 2.2 system that includes one DSDD 8-inch drive and one 48 tpi, DSDD 5¼-inch drive plus the same software provided with Disk Maker I. You can add the same options described for Disk Maker I, up to a total of four drives; a 10 Mb hard disk option is also available. Having reviewed the manual for the Disk Maker II (but not having used the equipment), I am reasonably confident that my remarks about Disk Maker I will be applicable to the operation of Disk Maker II as well.

The Disk Maker I controller uses the Western Digital 1795 controller chip, allowing the software to read or write disks in all 170-plus formats shown in the Table on page 101 (if you have the appropriate drives, of course). Disk Maker I cannot accommodate some disk formats such as Apple, NorthStar, Micropolis, Commodore, and Victor; these formats either are hard-sectored or depend on special tricks in the disk controller hardware.

When you have the optional 96 tpi 5¼-inch drive installed, the software allows you to set the 96 tpi drive as a read-only drive for a 48 tpi format; this allows you to transfer files from one 5¼-inch disk to another without having to do an intermediate copy. With the two 5¼-inch drives set to the same format, rapid disk duplication is possible using the DUPE utility program. New formats are added regularly, and New Generation Systems offers each upgrade to current users for only $50.

Programs

The Disk Maker I software includes these programs:

DMINSTAL.COM—sets user-determined and system-dependent options
DMSET.COM—loads Disk Maker BIOS into the system
DMFORM.COM—formats disks
WHATDISK.COM—determines exact format of IBM, Z-100, or Morrow disk
MC.COM—Mass Copy, a file copy program
DUPE.COM—rapid track-for-track copy program with optional verify
DMCOMP.COM—rapid track-for-track compare program
TOMS.COM—CP/M < = > PCDOS file exchange utility

Various public domain utilities such as SWEEP and D normally are provided on the distribution disk as space permits. Two or more versions of DMSET and DMFORM are provided to accommodate various hardware peculiarities.

The optional software is:

DDD.COM—disk alignment program used with special disks
DEC2CPM—utility to transfer files from DecMate II word processor to CP/M
WANG2CPM—utility to transfer files from Wang OIS word processor to CP/M

Requirements

Disk Maker I is designed to operate under standard CP/M 2.2 with an 8080, 8085, or Z80 CPU. New Generation Systems recommends using it with a standard CP/M system only, but I run it with CP/M 2.2 and ZCPR, version 1, without any problems. The controller board and Disk Maker software normally use ports 98 to 9F (hex), but New Generation Systems provides software and hardware modification instructions to relocate the ports to other addresses if you have an unresolvable conflict in your system.

Although a 64K system is recommended for Disk Maker I, it will run in as little as 48K. The more disks you attach to the Disk Maker, the more memory the software requires. When you use the TOMS utility, a smaller TPA means that you may not be able to handle a full directory on an MSDOS disk; my 57K system will handle the maximum. Double-density

Apricot CP/M-86	3"(SSDD)	306K	Intertec HeadStart	3"(SSDD)	372K
Access	(SSDD)	169K	Actrix Matrix	(DSDD)	348K
AMPRO	(SSDD)	188K	AMPRO	(DSDD)	386K
Balcones BRAVO	(DSDD)	346K	Bondwell Model 14/16	(DSDD)	346K
BYAD	(SSDD)	154K	Casio FP 1000/1100	(DSDD)	300K
Compuview CP/M-86	(SSDD)	193K	Cromemco	(SSSD)	81K
Cromemco	(SSDD)	188K	Cromemco C-10	(DSDD)	386K
Davidge 1024	(DSDD)	368K	DEC VT180	(SSDD)	169K
Digilog	(DSDD)	332K	Eagle I	(SSSD)	70K
Epson QX-10 CP/M	(DSDD)	378K	Epson QX-10 MF	(DSDD)	278K
Fujitsu Micro 16s	(DSDD)	314K	Heath H-37	(SSDD)	148K
Heath H-37	(DSDD)	304K	Heath w/CDR Systems	(DSDD)	380K
Heath w/Magnolia	(SSDD)	162K	HP 86/87/125	(DSDD)	248K
IBM PC (CP/M-86)	(SSDD)	154K	IBM PC (CP/M-86)	(DSDD)	314K
ICL PC1 10,30,31,32	(DSDD)	254K	Insight Dev. IQ-120	(SSDD)	146K
KayPro II	(SSDD)	191K	KayPro 4/10	(DSDD)	394K
Lobo Max-80	(SSDD)	164K	Lobo Max-80 35tk	(SSDD)	142K
Lobo Max-80 35tk	(DSDD)	296K	Microbee	(DSSD)	171K
Microbee	(DSDD)	386K	Molecular 9X	(DSDD)	338K
Morrow Micro Dec I/II	(SSDD)	186K	Morrow Micro Dec III	(DSDD)	384K
Morrow Micro Dec 11	(DSDD)	384K	MultiTech MIC-501	(SSDD)	168K
NCR Decision Mate V	(DSDD)	304K	NEC PC-8000 Series	(SSDD)	169K
NEC PC-8000 Series	(DSDD)	300K	Osborne	(SSSD)	90K
Osborne DD/Executive	(SSDD)	183K	Otrona Attache	(DSDD)	360K
PMC 101 (CP/M 3.0)	(DSDD)	386K	Sanyo 1000/1100/1150	(DSDD)	310K
SD Sales	(SSSD)	71K	SD Sales	(SSDD)	112K
Sharp 3540	(DSDD)	292K	SuperBrain	(SSDD)	162K
SuperBrain	(DSDD)	340K	Systel III	(DSDD)	346K
TeleVideo 802/3/6	(DSDD)	340K	TeleVid. 806 TurboDos	(DSDD)	396K
TI Professional	(SSDD)	154K	Toshiba T100	(DSDD)	254K
TRS-80 Mod I Omikron	(SSSD)	70K	TRS-80 Mod I Omikron	(SSDD)	131K
TRS-80 Mod III MM	(SSDD)	186K	TRS-80 Mod IV	(SSDD)	154K
TRS-80 IV Montezuma	(SSDD)	166K	Versa Floppy II	(DSDD)	278K
Xerox 820	(SSSD)	82K	Xerox 820	(SSDD)	155K
Zenith Z-100	(SSDD)	148K	Zenith Z-100	(DSDD)	304K
Zenith Z-100 (orig)	(DSDD)	304K	Zorba	(DSDD)	388K
ALTOS 586	96(DSDD)	700K	AMPRO	96(SSDD)	386K
AMPRO	96(DSDD)	782K	Analyzer	96(DSDD)	620K
Archives I	96(SSDD)	386K	Archives II	96(DSDD)	786K
Associate +	96(DSDD)	786K	CompuPro 10	96(DSDD)	772K
CompuStar 40	96(DSDD)	786K	DEC Rainbow	96(SSDD)	386K
Digilog DBS 16	96(DSDD)	780K	Direct OA1000	96(DSDD)	622K
Discovery	96(DSDD)	624K	Eagle II/IV	96(SSDD)	386K
Eagle III	96(DSDD)	762K	Epic	96(DSDD)	782K
Facit DTC	96(SSDD)	308K	Heath H-37	96(SSDD)	308K
Heath H-37	96(DSDD)	624K	Honeywell	96(DSDD)	632K
ICL Model 25	96(DSDD)	628K	ICL Model 35	96(DSDD)	778K
IMS (TurboDos)	96(DSDD)	772K	Intel iPDS	96(DSDD)	608K
Ithaca 525	96(SSDD)	346K	Ithaca 525	96(DSDD)	628K
Lobo Max-80	96(DSDD)	698K	MACSYM 150/350	96(SSDD)	308K
Monroe OC8820	96(SSDD)	306K	Monroe 2000	96(DSDD)	628K
MultiTech MIC-504	96(DSDD)	698K	NCR Work Saver	96(DSDD)	544K
Olympia People	96(DSDD)	618K	OSM Zeus 4	96(DSDD)	620K
Otrona	96(DSDD)	770K	Philips P2000C	96(DSDD)	628K
Philips 3000	96(SSDD)	294K	Pied Piper	96(DSDD)	776K

Table
Disk Maker I Formats

(Continued on next page)

Sanyo 1200/50	96(DSDD)	620K	Sanyo 2000	96(SSDD)	302K
Sanyo 4050	96(DSDD)	620K	TeleVideo 1603	96(DSDD)	706K
Vector 4-S	96(DSDD)	712K	Vector VSX	96(DSDD)	710K
ALTOS	8"(SSDD)	446K	CCS (256b)	8"(SSDD)	482K
CCS (1024b)	8"(SSDD)	596K	Colonial Data SB80	8"(SSDD)	596K
Colonial Data SB80	8"(DSDD)	1208K	Columbia 1800	8"(SSDD)	592K
CompuPro (1024b)	8"(SSDD)	596K	CompuPro (1024b)	8"(DSDD)	1192K
CP/M SSSD Standard	8"(SSSD)	241K	Delta Products 125	8"(DSDD)	1212K
Harris 1685-50MFT	8"(SSDD)	596K	Harris 1685-50MFT	8"(DSDD)	968K
Harris 1685-50MFT	8"(DSDD)	1196K	IBM 3740 TO Direct	8"(SSSD)	248K
IBM 5520 WP	8"(DSDD)	1132K	IBM 128b no skew	8"(SSSD)	241K
IBM 256b no skew	8"(SSDD)	482K	Insight IQ-120	8"(SSDD)	484K
Megaflex Apple	8"(SSDD)	558K	Molecular	8"(SSDD)	496K
NEC APC CP/M-86	8"(DSDD)	980K	S. D. Sales (128b)	8"(SSDD)	464K
S. D. Sales (256b)	8"(SSDD)	476K	Tarbell (128b)	8"(SSDD)	472K
TRS-80 II,12/6 P&T	8"(SSDD)	596K	TRS-80 II,12/6 P&T	8"(DSDD)	1210K
TRS-80 Lifeboat	8"(SSDD)	482K	TRS-80 Lifeboat	8"(SSDD)	596K
Vector 2800	8"(DSDD)	984K	Zenith Z-100	8"(SSDD)	482K
Zenith Z-100	8"(DSDD)	980K	ACT Apricot MSDOS	3"(SSDD)	314K
HP-150 MSDOS	3"(SSDD)	258K	Intertec H-S MSDOS	3"(SSDD)	395K
IBM PC	PCDOS1.1(SSDD)	156K	IBM PC	PCDOS1.1(DSDD)	315K
IBM PC	PCDOS2.0(SSDD)	175K	IBM PC	PCDOS2.0(DSDD)	354K
IBM PC-AT	PCDOS3.0(DSDD)	1186K	Kaypro 10	MSDOS (DSDD)	354K
Wang	MSDOS (DSDD)	354K	Zenith	ZDOS 1.1(DSDD)	315K
BurroughsB25 MSDOS	96(DSDD)	610K	DEC Rainbow MSDOS	96(SSDD)	384K
Eagle 1600 MSDOS	96(DSDD)	785K	Monroe MSDOS	96(DSDD)	712K
NCR D-M "M" MSDOS	96(DSDD)	790K	Otrona MSDOS	96(DSDD)	712K
Sanyo MBC MSDOS	96(DSDD)	792K	Tandy 2000 MSDOS	96(DSDD)	714K
Lomas Data MSDOS	8"(SSSD)	239K	NEC APC MSDOS	8"(DSDD)	1221K
Standard MSDOS	8"(SSSD)	246K	Zenith V1.1 ZDOS	8"(SSSD)	245K

NOTE: all formats 5 1/4" 48 tpi, CP/M-80/86 unless noted as:
3" - 3 1/2 inch format
8" - 8 inch format
96 - 96 tpi 5 1/4 inch format
SS = Single Sided; DS = Double Sided
SD = Single Density; DD = Double Density

Table
Disk Maker I Formats

8-inch formats and the IBM PCAT drive require at least a 4 MHz system clock; otherwise, a 2 MHz clock should be sufficient.

Installation of the Disk Maker I hardware is as easy as plugging the disk controller board into an empty slot in your S-100 bus motherboard and connecting the disk cable between the disk cabinet and the controller board. A menu-driven installation program allows you to customize some features of the system to your hardware and personal taste; the installation process takes only a few minutes.

How It Works

Disk Maker I is simple to use. To format a disk, type DMFORM to invoke the format program, then choose a format by number from the on-screen menu. You can also invoke DMFORM by making a format choice on the command line; the program proceeds without showing you the menu. You can return to the menu if you want to change your selection. DMFORM will verify a disk after formatting if you set a permanent option with DMINSTALL or specify verification on the command line when you invoke DMFORM.

The DMSET program gives you access to the Disk Maker disk drives. You use DMSET the same way you use DMFORM: you can choose from a menu or go directly to work by specifying a format on the command line. You can set each Disk Maker drive to a different format. Every time you warm boot, the Disk Maker software displays the disk formats you have selected.

DMSET installs a special BIOS under your normal CP/M system; you can run any CP/M program that will fit in the somewhat reduced memory left for applications. Even though

there is less memory, I have found it adequate for running any of my utilities, dBASE II or my text editor (which does bidirectional scrolling through a disk file). You probably will notice the memory reduction the most when you run a spreadsheet program or any other application that uses all available memory but does not scroll data to and from disk, the way many text editors do.

Because of the radically different structure of MSDOS files, you need a special program called TOMS (TO MSDOS) to access individual disk files on a MSDOS disk. You cannot run another program while using TOMS; if you want a CP/M application to access a MSDOS data file, you first must transfer the file to a CP/M format. TOMS is like a small part of the MSDOS operating system, providing the commands DIR, COPY, TYPE, and ERASE. TOMS cannot deal with subdirectories, so all files must be in the root directory on a MSDOS disk coming into the system; all files written by TOMS are in the root directory.

Performance

I have been using Disk Maker I for well over a year. The capabilities of the system are impressive, but even more impressive is the support New Generation Systems provides. I was an early user and, as you might expect, everything did not start out working perfectly. One annoying problem was caused by the routing of the data cable in the disk drive cabinet. The cable was picking up noise from the power supply, which gave me errors when reading the inner tracks of the 48 tpi drive. I solved the problem by changing the routing of the cable at the suggestion of New Generation Systems, but in the meantime, I received a replacement 48 tpi drive that had been specially aligned and certified. I have always received courteous and competent assistance anytime I have had a reason to call New Generation Systems with a problem or a question. You can expect the same treatment. (I was dealing with New Generation Systems as just another user, before I had thought of doing this review.)

The Disk Maker I hardware consists of off-the-shelf components integrated by New Generation Systems—nothing "razzle-dazzle" but obviously chosen carefully for quality and value. What makes the Disk Maker I package outstanding is the software and support. The programs perform well and are easy to use. The no-nonsense operator interface was designed for users who need to get things done quickly.

Although the software is not friendly in the current sense of having elaborate menus and on-line help facilities, each user prompt and program response is clear in its meaning. With one exception, which I will discuss later, all errors are trapped cleanly and appropriate error messages appear on the screen. The user interface is appropriate for the intended users of this package: software professionals, experienced computer users, and serious hobbyists with a need to transfer files between diverse disk formats. Operation is simple enough for a novice to use the system, too.

The Disk Maker I (and Disk Maker II) documentation is well organized and appropriate for the intended users. The *User's Manual* for Disk Maker I contains over 50 pages with a good table of contents in a three-ring binder. The table of contents is adequate for using the manual: no index is provided and none is needed. The manual, prepared with a word processor, is much cleaner in appearance than most manuals prepared this way. It is relatively easy to flip through the pages and spot whatever you are looking for.

New Generation Systems has assumed that you already know how to use CP/M and your computer. The manual will say "copy this file to another disk" rather than give a blow-by-blow description of how to use the file copy utility. There is a section titled "How To Run Disk Maker Without Reading The Manual" for the impatient and for the technical-minded a section on each program called "How xxxx Works" that explains the inner workings of the program.

I have validated Disk Maker's ability to format, read, or write disks in about 20 of the 170 available formats. I have used Disk Maker primarily to move files between an S-100 system and an Osborne I (double density) or between the S-100 system and a PCDOS system. Other than the read errors I encountered (now fixed), the Disk Maker hardware and software have performed flawlessly for me.

I have encountered a few instances where it has been necessary to bulk erase a disk and format it with Disk Maker I because, if I wrote on a disk already formatted by the receiving system, that system could not read the disk.

Although Disk Maker I is easy to run, one enhancement would be welcome. The Disk Maker I software assumes you know what disk format you are dealing with. If you tell it you are feeding it a disk of format X and then give it a disk of format Y, you may or may not get an error message; the program you run could go merrily along without complaint, producing either garbage files or a disk that the target system cannot read. The Disk Maker software cannot identify positively that you have inserted the correct disk format in the drive being written to because no standard means of identification exists for a blank formatted disk; even if you determine the sector size, sectors per track, and so forth, you still cannot deduce the directory size and other characteristics needed for positive identification.

WHATDISK enables you to determine which of several similar formats a disk might be, if you already know that the disk is for IBM PCDOS, Heath/Zenith Z-100, or Morrow Micro Decision. A generalized WHATDISK program to identify any disk format would be very useful. Such a program is impossible to write, however, for the above reasons.

What You Pay

The basic Disk Maker I, which includes the controller board, one 48 tpi, DSDD 5¼-inch drive, and the Disk Maker software is $1695. The 96 tpi, DSDD drive is an additional $395. The 8-inch drive and cabinet for Disk Maker I is $849. The 3½-inch drive costs $295, and the PCAT drive is $495. The word processing transfer utilities package is $295, and the disk testing software is $150 plus $40 each for the 48 and 96 tpi Dysan test disks. You also can purchase the Disk Maker I controller and software with no drives.

The basic Disk Maker II system is $3395 for the stand-alone 6 MHz Z80B system with one DSDD 8-inch drive, one 48 tpi, DSDD 5¼-inch drive, and CP/M 2.2; you only need to add a terminal. The option prices are the same as for Disk Maker I, and a second DSDD 8-inch drive costs $600.

Are these prices reasonable? It depends on what you need. Programs for the Kaypro, IBM PC, Morrow, and other computers that allow you to read, write, and format a number of 5¼-inch formats cost less than $100, but don't ask them to recognize an 8-inch disk drive, a PCAT drive, or a 3½-inch drive. At the other end of the spectrum are systems that claim less capability than Disk Maker II and cost over $5000. Disk Maker I and Disk Maker II offer outstanding versatility and performance for the money.

Circle **no. 66** on reader service card.

Circle **no. 53** on reader service card.

Circle **no. 73** on reader service card.

Conclusion

Disk Maker I is a solid product backed by a competent and cooperative company that operates as if it wants to remain in business for a long time. If you are a software developer, software distributor, user group, large office, or serious hobbyist with the need to exchange files between computers with various disk formats, you should seriously consider Disk Maker I or Disk Maker II as an answer to your needs.

AMPRO Little Board and Bookshelf Computers

Company: AMPRO Computers, Inc., 67 East Evelyn Ave., Mountain View, CA 94041 (415) 962-0230.
Price: $349
Circle Reader Service No. 105
Reviewed by Richard Conn

The heart of the AMPRO Bookshelf Computer® is the AMPRO Little Board®, a full-featured single-board computer that measures only 5¾-inch by 7¾-inch and is designed for mounting on the back of a standard 5¼-inch minifloppy disk drive. Although it is small in size, the AMPRO Little Board is big in features: the Little Board contains all of the hardware necessary to support a conventional CP/M environment.

The Little Board is based around a 4 MHz Z80A microprocessor and the associated Zilog Z80 family of support chips. On board this small computer is:

- A 4 MHz Z80A microprocessor
- 64K of dynamic RAM
- One 4K 2732-type EPROM
- A Z80A CTC (Counter/Timer Circuit)
- A Z80 DART (Dual Asynchronous Receiver/Transmitter)
- A parallel output port (based on D-type latches)
- A Western Digital 1770 Floppy Disk Controller Chip

The AMPRO Bookshelf Computer is an AMPRO Little Board computer housed in an attractive case with one or two 5¼-inch floppies, built-in power supply, two 25-pin EIA RS-232C connectors, one Centronics connector, and a connector for tagging on up to two more 5¼-inch floppies. An ON/OFF switch and a RESET switch are available on the front panel.

The AMPRO Little Board Computer

Hardware

The Little Board has two possible memory configurations, which depend on the setting of an EPROM-enable bit in the Board Control Register. Enabling the EPROM establishes the following memory configuration:

Address (Hex)	Memory Element
0000 - 0FFF	2732 EPROM
1000 - 7FFF	2732 EPROM duplicated on each 4K group
8000 - FFFF	RAM

If the EPROM is disabled, all memory from 0 to 0FFFFH is enabled as RAM.

A Z80 DART and RS-232C line drivers provide the serial input/output ports. The DART is an asynchronous device, providing for baud rates up to 38,400 on its A channel and 9600 on its B channel in the Little Board configuration. The Little Board does not implement all of the RS-232C signals on its two serial channels. Only the following signals are provided:

- Serial Data Output (RS-232C pin 3)
- Serial Data Input (RS-232C pin 2)
- Request to Send Handshaking Output (RS-232C pin 5)
- Clear to Send Handshaking Input (RS-232C pin 20)

Both channels of the DART are wired as DCE (Data Communications Equipment), which is the complement of DTE (Data Terminal Equipment)—the wiring of the RS-232C connector on computer terminals. Notably missing from this list is the Data Carrier Detect signal. You may use the Clear to Send signal in place of Data Carrier Detect.

The parallel port is an output-only port, wired in a Centronics-standard configuration. The Z80 CTC on the Little Board provides four counter/timers for use in conjunction with the hardware and software of the computer.

The Western Digital 1770 Floppy Disk Controller chip used in the Little Board design can work with just under 400K per disk on double-sided, 48 tracks-per-inch (TPI) drives and just under 800K per disk on double-sided, 96 TPI drives.

Software

The AMPRO Little Board is supplied with one copy of CP/M 2.2, which uses the ZCPR3 Command Processor in place of the CP/M CCP. Although only the ZCPR3 Command Processor, in a less than maximum configuration, is provided with the Little Board, you can obtain a more complete ZCPR3 implementation from Echelon (see below).

The ZCPR3 configuration distributed with the Little Board provides CP/M 2.2 compatibility while simultaneously presenting the following additional features:

- Multiple commands per line, separated by semicolons, enabling command lines like DIR;ERA *.BAK;DIR
- Automatic command search path that searches through directories in the following sequence when looking for COM files: (1) current disk, current user; (2) current disk, user 0; (3) disk A, current user; (4) disk A, user 0; (5) disk A, user 15; and (6) current disk, user 15
- Four-element shell stack, so shells other than the ZCPR3 Command Processor can act as the user interface to the system
- Several built-in commands, including: GET—load file anywhere in memory; JUMP—call subroutine anywhere in memory; and LIST—print file on printer
- Extended directory references over normal CP/M: D:—reference disk (e.g., TYPE B:MYFILE.TXT); U:—reference user area (e.g., DIR 5:); and DU:—reference disk and user area (e.g.,

ERA B7:*.*)

See articles in *Dr. Dobb's, Computer Language*, *Byte*, and other such magazines for more information on ZCPR3. The authorized agent for ZCPR3 is Echelon, Inc., 101 First St., Los Altos, CA 94022 (415) 948-3820. The ZCPR3 RCP/M and BBS are also available to you at (415) 489-9005.

The nine standard CP/M utility programs (ED, DDT, PIP, etc.) and thirteen additional utility programs are supplied with the Little Board. Included in this list is MULTIDSK, which allows the user to dynamically select one drive to support any one of over 40 different 5¼-inch floppy disk formats, and 48TPI, which allows the user to read 48 TPI disks on a 96 TPI drive.

One nice option that AMPRO gives the customer is that of purchasing the source code to several key programs at a reasonable price. The customer may purchase the Technical Support Software package, which includes the source files for MULTIDSK, the BIOS, and the BOOT.

Pricing

The AMPRO Little Board computer costs $349. This price includes the single-board computer and all of the software listed above. An extra $50 buys the Technical Support Software package.

The AMPRO Bookshelf Computer

Hardware

As mentioned above, the AMPRO Bookshelf Computer consists of an AMPRO Little Board, one or two 5¼-inch minifloppy disk drives, a built-in power supply, two 25-pin EIA RS-232C connectors, one Centronics connector, one connector for attaching up to two external minifloppy disk drives, and ON/OFF and RESET switches on the front panel. The Bookshelf is available in the following models:

Model	Configuration
121	One double-sided, 48 TPI drive (400K)
122	Two double-sided, 48 TPI drives (800K)
141	One double-sided, 96 TPI drive (800K)
142	Two double-sided, 96 TPI drives (1600K)

Software

All of the software provided with the AMPRO Little Board is also provided with the AMPRO Bookshelf Computer. In addition, a ZCPR3 shell designed specifically for use on the AMPRO computer is provided (in object code only). FRIENDLY is a screen-oriented tool that provides a directory of files to the user of the current disk and allows him or her to perform a variety of operations on the files displayed. By simply moving a pointer to a desired file and pressing a few keys, a FRIENDLY user can copy a file, delete a file, view a file on the console, print a file on the printer, run a program, mark a file for later delete or copy with a group, perform a sequence of commands on the file based upon a preprogrammed menu, and perform many more operations.

FRIENDLY is easy to use, usually requiring only a few minutes to learn, and it removes casual computer users from the CP/M command environment, relieving them of the necessity of learning how to use CP/M in order to use the computer.

A second software package provided with the AMPRO Bookshelf is T/MAKER, a product of T/MAKER Company. T/MAKER provides a screen-oriented, memory-based editor, text formatter, electronic spreadsheet, bar graph generator, and file management system in one integrated package. T/MAKER will run under conventional CP/M, CP/M enhanced by the ZCPR3 Command Processor, or FRIENDLY. Several articles have already been published on T/MAKER, and you can obtain further information by writing to T/MAKER Company, P.O. Box 6430, Falls Church, VA 22046.

Documentation

The documentation provided with the AMPRO Little Board is the *Little Board User's Manual*, a mainly hardware manual that describes the Little Board in great detail. It is well written and clear if you have a background in digital electronics.

No documentation on CP/M or the standard CP/M programs is provided. The first pages of the manual, however, give pointers to where to obtain this documentation. Also, no documentation other than a brief overview of ZCPR3 is provided. Again, the manual gives pointers to Echelon.

If you buy the AMPRO Bookshelf Computer, you get the *Little Board User's Manual* as well as documentation on FRIENDLY and T/MAKER. The documentation on FRIENDLY and T/MAKER is oriented to computer users having some limited experience with CP/M.

General Impressions and Comments

I have used the AMPRO Bookshelf for several months now. I started with the Model 122, which had two 400K, 48 TPI drives, and am now using the Model 142, which has two 800K, 96 TPI drives. I find its clean design, speed, and reliability to be excellent. Given my background in CP/M and digital electronics, I found all of the documentation on the Little Board/Bookshelf to be readable and understandable.

The Bookshelf is a nice computer for any user, especially with the FRIENDLY shell as a front end. Although it is a good choice for the first-time computer user, as with any computer, you must overcome a learning curve before you can expect effective use of the computer.

The AMPRO BIOS performs well. The speed of the disks with this BIOS is nice: the disks are faster than most 5¼-inch disk systems I have observed and many 8-inch disk systems I have used. At 800K per disk with the Model 142 (96 TPI drives), I find the disk capacity also to be quite reasonable.

Overall, the AMPRO Bookshelf and AMPRO Little Board are good computers, but there are a few drawbacks for some applications:

(1) Lack of documentation on CP/M and ZCPR3. However, AMPRO supplies pointers to where you can obtain such documentation, at additional cost.

(2) Limitation of serial I/O ports to only two serial RS-232C ports and one parallel port. Nevertheless, if your needs call for no more than a terminal, modem, and printer (as many popular computers are configured today), the AMPRO computers are adequate.

(3) Lack of hard disk support and expandability. The Little Board does not support hard disks or other such devices in its current form. You may buy, however, the Little Board SCSI/PLUS Adapter for $99. This adapter "piggy-backs" on the Little Board and allows you to attach a Winchester disk drive, slave processor and device boards (perhaps for I/O), and other facilities. This board, with some effort, may eliminate drawbacks 2 and 3.

I am excited about the Little Board and the Bookshelf for their applications in several environments. As a stand-alone personal computer with limited I/O resources and disk space, these computers are good choices. The SCSI/PLUS Adapter promises to expand their capabilities.

For embedded computer applications in which 4K of code is enough to run the application program, the Little Board provides a nice, inexpensive self-contained computer. The 4K 2732 EPROM can be programmed for the application, and 32K of RAM is available for data. The two RS-232C ports are there, as well as the parallel output-only port. No disks would be needed for this type of application. For embedded computer applications with disks, the AMPRO Little Board would "piggy-back" on a 5¼-inch 96 TPI minifloppy and feed off of the floppy's power supply. This would provide a very small embedded computer with 800K of disk and 64K of RAM.

As a slave computer to another computer, the AMPRO Bookshelf with its two disks could act as a background batch processor, communications controller (for a BBS), printer spooler, or other such intelligent and flexible slave device.

Overall, I like the AMPRO Little Board and AMPRO Bookshelf and intend to continue using my Bookshelf for a variety of slave computer and stand-alone applications (such as computer assistance for talks and presentations). I recommend it with some reservation to first-time computer users and with no reservation to embedded applications designers and more experienced computer users.

BetterBASIC, Version 1.1

Company: Summit Software, P.O. Box 99, Babson Park, Wellesley, MA 02157 (617) 235-0729
Computer: IBM PC or close compatible with minimum 192K and one disk drive
Price: BetterBASIC $199; 8087 Math Module $99; Runtime System $250
Circle Reader Service No. 107
Reviewed by Matthew Trask

Among the reasons that Summit Software gives for purchasing BetterBASIC are full 640K memory support, separately compiled program modules, language extensibility, window support, 8087 support, and incremental compilation. Although these reasons would be persuasive in the case of another programming language, I question whether they will induce many programmers to shell out $199, the cost of BetterBASIC, when some form of GW BASIC (aka BASICA) is provided at no charge with most IBM type computers these days and an excellent Pascal compiler can be purchased for less than fifty dollars. In order to help programmers make a decision on this purchase, I will discuss the unique features of this programming system, benchmark its performance, and examine the difficulties of converting programs from GW BASIC to BetterBASIC.

Installation

Although the package can be run right out of the box by using its default configuration, you will probably want to configure it to suit your hardware environment. B.COM is the executable portion of BetterBASIC and it uses the file B.CNF to determine which of the other modules to load for operation. Table 1 (page 110) is a listing of all the modules that are provided on the BetterBASIC disk and Table 2 (page 110) shows the default configuration. An ASCII editor can be used

to add or delete modules and create new configurations. For example, users that don't have a graphics card may wish to delete GRAPHICS.IBM in order to free up memory for program use. I replaced FILE.DOS with FILE2.DOS in order to take advantage of DOS 2.x subdirectories and added SYSCALL.IBM in order to use the SHELL command and BIOS/DOS calls. You may specify the use of alternative configurations when starting the program by typing B-MYCONFIG/c where MYCONFIG is the name of the optional configuration file.

A Tandy 2000 version of the program is also available. The modules with the extension .IBM are all replaced with equivalent modules that end with .TDY.

Startup and Operation

After a sign-on message and copyright notice you will see the first major difference between BetterBASIC and GW BASIC: instead of the usual "60891 Bytes free" message, a 512K machine will display "Left:260304 bytes." All available system memory is used as BetterBASIC's workspace. This allows much larger programs to be written. BetterBASIC supports most of GW BASIC's commands, including the on-screen editing keys that are used on the IBM PC. Table 3 (page 111) contains a complete listing of GW BASIC commands that are different or not supported and a list of new commands that are found in BetterBASIC.

Every statement that you enter is checked for syntax and then compiled. If you make a mistake when entering a program line, an error message will be given immediately. For example:

```
40 PRINT (1 + 2 * 3
---------------^
" ) " Expected
```

If the statement is entered correctly, it is compiled to virtual-machine code rather than being interpreted at runtime, thus giving a large speed improvement over the GW BASIC interpreter.

As in Pascal and C, all variables are declared at the beginning of the program. The exception to this is the AUTODEF command, which allows automatic declaration of a variable by the first assignment of a value. AUTODEF defaults to ON, but if you turn it off, the compiler will catch misspelled variable names. The data types that are supported in BetterBASIC are byte, integer, real, string (up to 32,767 characters), array, pointer, and structure. Structures are identical to Pascal's record data type and may contain any other data type. Arrays can contain byte, integer, real, and string data as well as structures and other arrays, thus allowing arrays of arrays, arrays of structures, and structures of arrays.

A BetterBASIC programming session superficially resembles one in GW BASIC in that lines are numbered and most of the command keywords are identical. When saving to disk, however, you must specify the file's extension because .BAS is not assumed by BetterBASIC. At the

B	COM	35312	1-03-85	1:00a
B	DEF	27020	1-03-85	1:00a
B	CNF	142	1-03-85	1:00a
MATH	BCD	8112	1-03-85	1:00a
CONSOLE	IBM	21392	1-03-85	1:00a
MAIN		3840	1-03-85	1:00a
FILE	DOS	19056	1-03-85	1:00a
GRAPHICS	IBM	10512	1-03-85	1:00a
PLAY	IBM	3136	1-03-85	1:00a
EVENT	IBM	7744	1-03-85	1:00a
FILE2	DOS	21072	1-03-85	1:00a
CHAIN	MOD	2704	1-03-85	1:00a
SYSCALL	IBM	3520	1-03-85	1:00a
COMM	BAS	8484	1-03-85	1:00a
PLIST	BAS	2542	1-03-85	1:00a
SECTOR	BAS	6702	1-03-85	1:00a
STRUC	BAS	2980	1-03-85	1:00a
SHELL	CNF	175	1-03-85	1:00a
CHAIN	SIZ	11	1-03-85	1:00a
READ	ME	126734	1-03-85	1:00a
	20 File(s)		1024 bytes free	

Table 1
Directory listing of the BetterBASIC diskette showing the many modules that make up the Better BASIC system.

```
MODULES=MATH.BCD
MODULES=CONSOLE.IBM
MODULES=MAIN
MODULES=FILE.DOS
MODULES=GRAPHICS.IBM
MODULES=EVENT.IBM
MODULES=PLAY.IBM
STATUS=ON
```

Table 2
The default B.CNF configuration file.

end of a session you can exit to DOS with the SYSTEM command or by typing BYE.

Unique Features of BetterBASIC

The best feature of BetterBASIC is the use of true procedures with variables of local scope that exist only inside the procedure. Global variables can be used by declaring them as EXTERNAL in the procedure and providing the actual declaration at a level outside the procedure. A procedure can be declared as RECURSIVE if its name is declared as EXTERNAL inside the procedure. GOSUB and RETURN are both supported, but as you gain experience with BetterBASIC, they will probably fall by the wayside in favor of these Pascal-like, parameterized procedures. In addition to passing arguments by value or reference, you may specify optional and/or default arguments.

One particularly distinctive aspect of BetterBASIC is the ANY ARG declaration. This allows a procedure or function to determine what type of data was passed to it at runtime with the following functions:

TYPE( ) extracts the type of datum

DIM( ) extracts the dimensions of a datum

SIZE( ) extracts the size of a datum

For example, A=TYPE(FOO) will assign the following values to A:

0	if FOO is a byte
1	if FOO is an integer
2	if FOO is a real
3	if FOO is a string
4	if FOO is a structure
10 +N	if FOO is an array of above type N

If TYPE(A) returns 3 for string you

GW BASIC Statements Not Supported in BetterBASIC

BLOAD	BSAVE	CDBL	CSNG	CVD
CVI CVS	CONT	DEFDBL	DEFINT	
DEFSTR	DEF FN	DEF USR	EQV	ERASE
ERL FIELD	FRE	GET	IMP	
MERGE	MKD	MKI	MKS	MOTOR
ON PEN	ON STRIG	PEN	PMAP	PUT
RESUME	STICK	STRIG	TROFF	TRON
USR VARPTR	VIEW	WEND	WINDOW	

New BetterBASIC Statements

ANY ARG	ASH	AUTODEF	BYE
BYTE	BYTE ARG	BYTE ARRAY	BYTE ARRAY STRUC
BYTE PTR	CHECK	CODE	CODE?
COLOR BORDER	COMPRESS	CONSTANT	DEFINE WINDOW
DEL$	DO . . END DO	DO IF	DO UNTIL
DYNAMIC	EDIT proc	ERROR	EXIT
EXIT n LEVELS	EXTERNAL or EXT	FRAME WINDOW	HEADER
INPUT FROM	INS$	INTEGER or INT	INT ARG
INT ARRAY	INT ARRAY STRUC	INT FUNCTION	INT PTR
INTERRUPT	INTERRUPT PROC	KEYWORD ARG	LINE$
LIST ALL	LIST ARGS	LIST PROCS	LOWER$
MAIN	MAKE MODULE	MAKE PROGRAM	OFFSET
ON INTERRUPT	PRECISION	PROCEDURE or PROC	PUBLIC
READCHR	READCHR FROM	READLINE	READLINE FROM
REAL	REAL ARRAY	REAL ARRAY STRUC	REAL ARG
REAL FUNCTION	REAL PTR	REPEAT	REPEAT IF
RESTORE SCREEN	RESULT=	ROT	SAVE MODULE
SAVE SCREEN	SEG	SET	SH
SIZE	SPAN	STRING or STR	STR ARG
STR ARRAY	STR ARRAY STRUC	STR FUNCTION	STR PTR
STRUCTURE	UPPER$	WHILE . . DO	XREF

Table 3
A comparison of BetterBASIC and GW BASIC Keywords

can determine its allocated size (not length) with SIZE(A). If FOO is a structure, DIM(A) will return the number of fields and TYPE(FOO(N)) will return the type of field N. This flexibility of optional/ambiguous parameter passing can be a tremendous improvement over Pascal's rigid requirements.

BetterBASIC has a concept called Procedure Families in which more than one procedure or function can have the same name with the only distinction being the parameter list. This allows BetterBASIC to select which one to use based on the type of argument that is used. As an example, the two procedures that follow can be used to give informative error messages on integer input, rather than GW BASIC's cryptic "Redo from start."

```
PROCEDURE:IntegerInput
INT ARG Value
10 INPUT Value
20 PRINT "Your integer is ",
   Value

PROCEDURE:IntegerInput.a
STRING ARG SomeString
10 PRINT SomeString, "is not a
   valid integer, please try again"
```

In this example, IntegerInput.a will be used any time a valid integer is not entered when IntegerInput is called. This technique can also be used to assign additional functionality to BetterBASIC's reserved words.

Functions are similar to procedures in that they share all the same data types. The difference is that they are used in expressions to return a value that is used in the expression. The last statement of a function must be RESULT= and the value that is to be returned.

Both procedures and functions can use KEYWORD ARGuments—a concept that is similar to Pascal's user-enumerated data types. If an argument is declared as

```
KEYWORD ARG:Color
   \RED\GREEN\BLUE
```

then the procedure will accept either RED, GREEN, or BLUE as valid data for Color.

BetterBASIC has a full complement of flow-controlling block structures. In addition to the usual FOR..NEXT, IF..THEN, and WHILE..REPEAT, BetterBASIC has DO UNTIL..REPEAT, DO..REPEAT, DO n TIMES..REPEAT, DO..END DO, DO..REPEAT, DO IF..REPEAT IF, and more. GOTOs are implemented in a way that could placate even Dijkstra—they cannot be used to enter or exit a control block. The only way out of a control structure is via the EXIT statement, and then only to the statement that immediately follows the block.

Variables can be declared as absolute by adding a segment and offset after the declaration. BYTE ARRAY(4000):VideoBuffer [&hB000:0] is an absolute reference to the IBM's 2000 character monochrome video refresh buffer. This technique can be used to access the BIOS communication area in low memory for equipment determination.

The BetterBASIC language can be extended by adding your own procedures and functions to the language as keywords. The MAKE MODULE command will convert all procedures and functions currently in memory to a module that can be added to the B.CNF configuration file. Like C language function libraries, useful procedures can be distributed to other BetterBASIC users as modules with a good degree of source code protection because the source code cannot be derived from a module. Summit Software is currently selling an 8087 module and expects to offer an interface module for the SoftCraft Btrieve system sometime this spring.

Programs can be prepared for standalone execution with the MAKE PROGRAM command and the optional Runtime System. The Runtime System consists of RUN.COM, RUN.$$$, and MKEXE.COM. RUN.COM will execute any program that has been prepared with MAKE PROGRAM and MKEXE.COM appends the RUN.$$$ runtime library to a program file for stand-alone execution.

Assembly language support is pretty straightforward with a very good example given in Appendix G of

the manual. The keyword CODE is used instead of BLOAD in order to load an assembler module into memory at runtime. CODE? can be used to report the absolute load address if debugging is necessary. The assembly language procedure is then invoked by CALL procedurename(arglist) just as in GW BASIC. An assembly language module can contain one or more procedures with a Symbol Table at the beginning that defines the entry point for each procedure in the module. The excerpt from the BetterBASIC manual in the Listing (page 116) will give the flavor of assembly language interfacing.

Also worthy of note are the XREF command and the debug window. XREF generates a symbol cross-reference of all user-declared identifiers and the line numbers on which they appear. Advanced programmers may wish to use the DOS debug program to test programs that are developed in BetterBASIC. If they start with DEBUG B and then execute BetterBASIC inside DEBUG, the [Ctrl-PgUp] keys will exit to the debug prompt.

Current Version

The current version of BetterBASIC is 1.1. I received the upgrade from 1.0 to 1.1 during the review. New features supported include CHAINing and CALLing modules as overlays, communication between overlays via disk, SYStem calls for access to BIOS and DOS functions, and a SHELL command that can be used to execute programs (such as COMMAND.COM) from within BetterBASIC. The new XMEM keyword allows data structures such as arrays up to the limits of system memory, not just 64K as in v1.0. Also, the release notes describe bug fixes that were applied to this new version.

Summit Software informs me that all registered users will be provided with free updates to v1.1, with a copy of the new documentation on the disk and an option to purchase printed documentation for $15. I also received test versions of the MATH1.BNY and MATH2.BNY (single and double precision binary math) that are significantly faster than MATH.BCD, but less precise. By the time you read this, both modules should be included in the standard BetterBASIC system in addition to the BCD math module.

Benchmarks

I compared the performance of BetterBASIC and GW BASIC by running Rich Malloy's four BASIC benchmark programs that I downloaded from the *Byte* bulletin board. These tests separate I/O from computation to give fairly accurate representations of speed. With one exception (see Table 4, page 114), BetterBASIC was the better performer by far. The exception was in the floating point calculations, where the BCD math module was about 30% slower than GW BASIC. The new single precision binary math module ran about twice as fast.

Round-off error can be of concern in business and financial programming. Although MATH1.BNY showed no error in this benchmark (compared to GW BASIC's −1.788 E−07 error), it is probably well worth the slower speed of MATH.BCD to have guaranteed zero error in financial calculations.

The Runtime System, unlike the BASCOM compiler, will not show an improvement in execution speed. This is because BASCOM generates native 8086 code while MKEXE and RUNCOM only provide a stand-alone BetterBASIC environment to run the virtual-machine code that is produced by the MAKE PROGRAM command.

These benchmarks were run on an IBM PC with 512K of RAM, 10Mb hard disk, and a monochrome display. I have also successfully run BetterBASIC on a Leading Edge PC/XT, a PCjr, and a Hyperion PC with no difficulty.

GW BASIC Program Conversion

To put it bluntly, there is no easy conversion from GW BASIC to BetterBASIC. There is a simple procedure outlined at the end of the manual that involves editing an ASCII version of the original. The keywords SOURCE and ENDFILE are added at the start and end of the file for compatibility with BetterBASIC's LIST ALL ASCII storage format. In addition, variable

declarations must be made at the start of the file. If you don't declare the variables, they will be defined on use, but all integers will be defined as reals with the resultant loss of speed and precision that comes from the use of the real data type.

I was able to convert with no difficulty trivial programs such as the *Byte* benchmarks, and one fairly lengthy musical program came over without much modification. However, large programs such as PC-TALK or RBBS-PC would be easier to rewrite from scratch with BetterBASIC than they would be to convert. This is because of major philosophical differences in the use of procedures and the use of structures for file I/O rather than FIELDs in an I/O buffer. Conceptually, it would probably be easier to translate a Pascal or C program than an existing GW BASIC program.

Refer to Table 3 for a complete summary of the differences between the two BASICs.

The Documentation

If there are two words that describe the BetterBASIC documentation, they are massive and thorough. The manual is an IBM-style 5½ by 8-inch slip-cased binder with over 550 pages! It is broken up into seven sections: General Information, Introduction to BetterBASIC, Quick Reference, Syntax Visuals, Appendices, and two new additions for v1.1—Chain/Overlays and Syscalls.

The syntax visuals are comparable to the IBM BASIC manual with one or more pages in alphabetical order to describe each command and show examples of proper usage. The introductory chapter is very well done with an excellent tutorial. It is geared to beginners, but will provide a refresher for experienced BASIC programmers. It includes in the lesson on recursion a discussion of static versus dynamic storage of variables that will be useful to many microcomputer programmers. The presentation of variable scope and side effects in the procedure section will be helpful to those programmers whose only previous experience is with other BASICs. Appendices include an ASCII code table, keyboard scan codes, interrupts, character sets, module usage, module defined statements, use of Assembler, error codes, and GW BASIC conversion hints.

User Support

Because Summit Software is still a small company, any phone calls in search of support will be handled by one of the program's developers. This kind of direct access can be wonderful, especially when problems involve the more complex aspects of the system. There is no charge for phone support for registered users and Summit can be reached during business hours.

Complaints

As in all new products, there is still room for improvement. My main gripe has to do with errors in the manual. Until I received my v1.1 upgrade, I could find no reference to the PUBLIC keyword that is necessary for preparing to MAKE MODULEs. There also seems to be some confusion when conceptual words from other languages are used interchangeably with BetterBASIC keywords; structure and record are used in this fashion, although record is never defined to mean the same as structure.

Some of the error codes that the compiler can generate are similar to the warnings of other compilers. I'd like to see the compiler add some of the missing parentheses that it can detect, rather than just notifying the programmer. Also, you cannot specify a program on the command line to be executed by BetterBASIC. This precludes the use of B SETCLOCK to read your clock/calendar in an AUTOEXEC.BAT file.

There is no provision made for tracing program execution, as TRON/TROFF are not supported. I hope the people at Summit are listening and provide a trace that, unlike GW BASIC's primitive version, can also show variable contents, not just line numbers.

My final complaint has to do with price—there is a hidden cost that a potential developer must keep in mind. If you plan to market the software you develop with BetterBASIC, you will need to purchase the Runtime System for an additional $250 so your customers will be able to run your program without the necessity of purchasing their own copy of BetterBASIC.

Conclusions

So who will find this package useful?

	BetterBASIC	GW BASIC
BENCH1.BAS—write 64K bytes out to a file	14.6	43.3
BENCH2.BAS—read 64K bytes in from a file	10.4	29.1
BENCH3.BAS—5000 floating point operations	92.9(BCD) 37.2(BNY)	69.4
BENCH4.BAS—calculate all primes from 1 to 7000	35.2	208.1

Table 4
Performance Benchmarks (All times in seconds)

Probably not the C wizards and Forth gurus of the world. They will feel restrained by the line numbering and the lack of mysterious, syntactic constructions. Pascal programmers will feel right at home with procedures that have local variables and statements like READLINE. I think this package will find its place mostly among programmers who are currently developing vertical market (i.e., business) software in BASIC.

A large percentage of software that is created for vertical markets, such as insurance agencies, medical practices, video rental clubs, and accounting firms, is written in some form of the BASIC language. One reason that is often given for this disproportionate popularity is that BASIC is available in some form on most personal computers, thus providing a greater market when the software is ported to a new machine. However, because of the widespread availability of Pascal and C compilers for micros in recent years, I think this reason is less important than the fact that much vertical market software is not developed by trained professional programmers. Often an accountant/car salesman/real estate broker/farmer who has learned a little BASIC on a personal computer realizes a need for some industry-specific piece of software. This combination of industry expertise and programming experience can result in a very successful program.

In the words of Ivar Wold, Summit's president, these people are faced with the "Pascal Dilemma"—they intend to learn Pascal or C but haven't found the time for it yet. With this package, a BASIC programmer will be able to write better, more maintainable code without the learning curve that is associated with learning a new language from scratch. After using BetterBASIC, it will be easy to pick up Pascal with very little effort, as all of Pascal's concepts can be found in BetterBASIC. However, BetterBASIC may provide just the excuse never to have to learn a new language again.

DDJ

```
  Sample assembly language interface to a BetterBASIC program
        org     0

;--- define the Symbol Table

sym1    db      sym2-$              ;link to next symbol table entry
        db      'EXCHANGE'          ;name of the Procedure
        dw      0                   ;fill word ( MUST be here !!)
        dw      header1             ;pointer to Procedure header

sym2    db      sym3-$              ;link to next symbol table entry
        db      'FILL'              ;name of the Procedure
        dw      0                   ;fill word ( MUST be here !!)
        dw      header2             ;pointer to Procedure header

sym3    db      0                   ;Symbol Table Terminator

;--------------------------------

        even                        ;headers must be on
                                    ; WORD boundary

header1 dw      154h                ;MUST be 154h
        dw      exchange            ;pointer to actual asm procedure
        dw      2                   ;number of Arguments
        dw      1                   ;type of argument 0 (Integer)
        dw      1                   ;type of argument 1 (Integer)

header2 dw      154h                ;MUST be 154h
        dw      fill                ;pointer to actual asm procedure
        dw      2                   ;number of Arguments
        dw      3                   ;type of argument 0 (String)
        dw      1                   ;type of argument 1 (Integer)

;--------------------------------

; These are the procedures proper

exchange        proc far            ;MUST be a far proc
        .
        .                           ; the EXCHANGE code
        .
exchange        endp

fill            proc far
        .
        .                           ; the FILL code
        .
fill            endp
```

End Listing

Circle no. 134 on reader service card.

Circle no. 86 on reader service card.

Circle no. 19 on reader service card.

by Ray Duncan

68000 Square Root Routine

Jim Cathey of Spokane, Washington, writes: "To help [eliminate] the dearth of 68000 programming goodies you were mentioning, here is an integer square root program. It takes only about 10–11 times as long to execute as a divide instruction. I found this algorithm a few years back in *EDN* .... I thought it was a nifty trick, and saved a clipping for when I needed it. Two muls instructions, one add.l, and this subroutine implement a true 16-bit vector length subroutine, in less than 2000 cycles (250 microsec. at 8 MHz)!

"The original program (and my first attempt) had a bug, in that the intermediate variables were half the size of the input data. This works until the last trip through the loop, where you may have overflow errors that could ruin the accuracy. For speed, the main loop could be unrolled using word intermediates for the first 15 passes, and long on the last loop-equivalent. This should save some time if it's a problem. Of course, if speed is really a problem, one of the approximations for vector length you presented a while back would be many times faster." See Listing One (page 122) for the source code for Jim's 68000 square root routine.

MacFeedback

Jim Howell of San Jose, California, a regular correspondent to *DDJ*, writes: "I am writing regarding your recent rather disparaging comments (*DDJ*, December 1984, #98) on the Macintosh. I believe it is much too early for you to be forecasting the demise of the Mac. So 'only' 200,000 Mac's were delivered in its first year. How many IBM PC's were delivered during its first year? And have you forgotten that even a year after the IBM PC came out, people were still complaining about the lack of software for it? As for comparing the Mac's 'lackluster sales' with the Lisa, I would guess that there are already more Macs than Lisas.

"One point I will agree on is the difficulty of developing software for the Mac. If Apple really wants lots of software for it, they need to make the required technical documentation available (and at a reasonable price). Development tools (compilers and assemblers) need to be able to run using a single Mac. There was an ad in *Infoworld* recently for a Mac assembler 'that requires no other hardware or software.' I don't know much about the availability of other language processors for the Mac, though I suspect (and hope) that most of them will be usable with only a single Mac. As for 'weak, bug-ridden' high-level languages, again consider the IBM PC. That's probably a fair description of many of the language processors that were available for the IBM PC during its first year. IBM's Pascal compiler, for example, is atrocious.

"I also agree that a 68000 assembler should be able to run with 128K and one disk drive, but comparing that to the DRI 8080 assembler that ran in 32K is very unfair. 8080 assembly language is much simpler than that of the 68000. The 68000 has more addressing modes, more instruction formats, and its assembly language has a more complex syntax. Also, the DRI 8080 assembler must have been a bare-bones, absolute assembler. A bare-bones 68000 assembler could be written to run on a 64K CP/M system (but probably not on a 32K system). A similar assembler should run, therefore, on a 128K Mac. However, I would assume that Apple does not want to put out just a bare-bones assembler. If, for example, the Apple assembler generates object code that can be linked with separately compiled or assembled modules, this would add considerable complexity to the assembler.

"While a Mac assembler should be able to run in 128K, I don't think it's a big problem if it does not. Anyone doing serious software development on the Mac will have to have 512K (the next step after 128K) and two disk drives, even if some of the tools run in 128K. After all, how many developers of IBM PC software have only 128K in their machines? And even fewer have only one disk drive.

"I will be like the other readers who requested 68000 material and not submit any 68000 code. Like most of these readers (probably), I am interested in the 68000, but do not actually have a 68000 system. Therefore, I also do not have any 68000 code to submit. With the attitudes expressed in your column about the 68000 (and about Intel's 80286), one wonders whether you would want to include any 68000 code or material anyway.

"Finally [in reference to another column on identifying IBM PC environments], location F000:FFFEH on my Corona contains 0FEH, just like the PC/XT. Apparently, they want software to think it's running on an XT, just in case a hard disk happened to be attached (yes, I do own an IBM compatible, though I like my 6809 system better). The AT&T computer that we use at work has 00 (zero) at that location (the date in the ROM is 05/03/84)."

This letter from Jim neatly illustrates the polarization that the Mac has induced within the programming

community. There is a certain class of people who are dazzled by Apple's "insanely great" publicity and will forgive the Macintosh anything. There is another group, among whom I guess I must number myself, who find Apple's pretentious advertising and grandiose claims a little ridiculous, considering that almost everything that's good about the Mac's user interface was "borrowed" directly from the Xerox Star that has been around for ten years. Speaking of pretentious, see the recent *Playboy* interview with Steve Jobs for some amusing pronouncements, especially about us programmers over 30 years old.

The fact is that everybody would find the sales of 200,000 Macs in the first year to be very remarkable, if Apple hadn't spouted off all those projections about how it was going to build a Mac every 15 seconds in their new super MacFactory. To argue that the Mac has already sold more machines than the Lisa is not what you'd call a devastating rejoinder, since Apple has already thrown in the towel on the Lisa and reincarnated it as the Macintosh XL.

As for the Macintosh assembler, I feel that I'm on very solid ground with my previous comments, having written several assemblers myself. The fact that the instruction set of the 68000 is powerful and extensive cuts both ways: of course it means that you have to worry about assembling more complex syntax, but it also means that you have a much more powerful language in which to *write* the assembler. Unless, of course, you are so bull-headed as to insist on writing your assembler in a high-level language like Pascal, which is completely unsuited to the job. Then you *do* end up with monstrosities like the Apple Macintosh Assembler or the Microsoft 8086 Macro Assembler. The proof of my contention has been conveniently provided by the product "MacASM" from Mainstay of Agoura Hills, California. This is an integrated full screen editor, macro assembler, and resource compiler for the Macintosh that runs nicely in 128K and costs only $125.00. "MacASM" is not copy protected and supports multiple drives including hard disks. Mainstay can be reached at (818) 991-6540.

MSDOS Device Drivers

Stan Mitchell of San Jose, California, writes: "I thought I would pass along a tool that comes in handy when working with MSDOS device drivers. It depends upon a useful property of the NUL device driver. Unlike other devices, it is embedded in the DOS module (IBMDOS.COM for the PC) and its device header is at the head of the device chain. By using the next device pointer in successive headers, the complete chain can be revealed. It is interesting that if any installed devices are present, the NUL header points to the first of these. When installed devices are exhausted, the link continues with the resident devices (contained in IBMBIO.COM for the PC).

"The only 'trick' in the program is using the OPEN function to return the device header pointer. This is an undocumented feature of MSDOS."

Readers, please note that the format of the reserved area in the file

control block is different under MSDOS 2.0 than it is under version 3.0. Stan's C program and the assembly language source for the _peek function are provided as Listing Two (page 122) and Listing Three (page 125).

Microsoft Assembler Bug of the Month

Gregor Owen of Port Jefferson Station, New York, writes: "I don't know if anyone's still counting, but here's my contribution for Microsoft Assembler bug of the month. In the assembly below, the code es:lodsb produces an error; however, the code below it, mov ax,es:[si+1], produces an error too. If you comment out the es:lodsb, *both* errors disappear—so superior to the old-fashioned kind of assembler, where frequently the removal of defective code only removes errors in the vicinity of the defect.

"At this point, I don't really know if es:lodsb represents a legal operation. I don't happen to have an Intel book at this location, and since every 8086/88 instruction has a unique set of addressing modes, it's difficult to get any kind of feel for what's allowed. Needless to say, the IBM/Microsoft assembler manual is not helpful. It's true that if one comments out es:lodsb and assembles lods byte ptr es:[si+0] both errors disappear, but I don't think anybody who's used the Microsoft assembler for any short while would take any comfort from that.

"And while we're on the subject, notice the expressive elegance and beauty of the phrase

```
lods byte ptr es:[si + 0]
```

typical of the language Intel has provided for us. The si+0, I concede, is something I've been doing ever since I managed to assemble some kind of complicated expression that had [si] in it—producing code that referenced [si+0FFFFH] and didn't generate an assembler error. Of course I probably don't have to use si+0 in every expression, but who wants to spend the time to find out? That's the beauty of the language: every programmer, in order to produce stuff that works at all, will develop a private dialect containing much myth and legend, thus making the source even more incomprehensible than it is anyway.

"My personal theory about 8086/88 assembly language is that what we have here is a marketing coup on the part of Intel. They had a ridiculous processor, with the traditional inept Intel approach to registers and addressing. Somehow, they sensed it would be awfully embarrassing to expose this thing in all the stark simplicity of their 'lxi h,7'-type assembler [for the 8080 among other processors]. So they commissioned the finest, most convoluted, Pascalized minds in the software industry to come up with an assembler so grand and structured and incredibly intricate that no one would ever notice how awful the thing was for which it was assembling. Judging by the results, they seem to have succeeded.

"And those of us who labor in the vineyards of the information revolution can't be all too high-horsey about this: after all, the Intel/Microsoft language makes it almost impossible for those pesky amateurs to figure anything out, thus preserving the world of 8086/88 assembler for us, the elect: people paid to spend the necessary endless hours figuring out how to use the language and avoid the Microsoft bugs. A sort of 'programmer's employment project' for the new age."

Gregor's program is provided as Listing Four (page 126). My uneducated guess, in this case, is that the Microsoft Assembler interprets the es: portion of the statement es:lodsb as the definition of a label named es. It then gives you the error message "symbol already different type," since the name es is already hardwired into the symbol table as an assembler directive. All the other errors follow from the ensuing type conflicts.

Fun with your PC/AT

Assemble the following instruction sequence into a short assembly language program and execute it on the IBM PC/AT of your choice:

```
mov  bx,0ffffh
```

```
mov ax,[bx]
```

The system dies immediately. The same sequence of instructions will run just fine on an 8086 or 8088-based machine. What is happening here?

It turns out that on the 80286, accessing a word operand at offset 0FFFFH, i.e., a segment wrap, causes a hardware interrupt 13H. Alas (as Jerry Pournelle would say), this interrupt is already used by the IBM PC ROM BIOS for the disk driver. So when your program accidentally generates a segment wrap fault, it traps to the disk driver with nonsense parameters in the registers and probably writes garbage all over your nice hard disk. This kind of phenomenon, when we move a previously happy program from an IBM PC to a PC/AT and watch it go to heaven, helps us while away those rainy afternoons.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 196.**

16-Bit Toolbox (Text begins on page 118)

Listing One

```
*   Integer Square Root (32 to 16 Bit)
*
*   (Exact method, not approximate)
*
*   Call with:
*           D0.L = Unsigned number
*
*   Returns:
*           D0.L = SQRT (D0.L)
*           D1.L <> 0 if not exact root
*
*   Uses:
*           D1-D4 as temporaries ---
*           D1 = Error term
*           D2 = Running estimate
*           D3 = High bracket
*           D4 = Loop counter
*
*   Notes:
*           Result first in D0.W, but is valid in longword.
*           Takes from 1480 to 1832 cycles (including RTS).
*           (Word version is from 548 to 660 cycles).
*
lsqrt       move.w    #15,d4     ; Loop count (bits-1 of result)
            moveq     #0,d1      ; Result in D1
            moveq     #0,d2
sqrtl       asl.l     #1,d0      ; Get 2 leading bits at a time and
            roxl.l    #1,d1      ; into Error term for extrapolation.
            asl.l     #1,d0      ; (Classical method, easy in binary)
            roxl.l    #1,d1
            asl.l     #1,d2      ; Running estimate * 2
            move.l    d2,d3
            asl.l     #1,d3
            cmp.l     d3,d1
            bls       sqrt2      ; New error term > 2* running est.?
            addq.l    #1,d2      ; Yes, we want 1 bit then.
            addq.l    #1,d3      ; Fix up new error term.
            sub.l     d3,d1
sqrt2       dbra      d4,sqrtl   ; Do all 16 bit-pairs.
            movel     d2,d0      ; Returns answer in D0.W
            rts
```

End Listing One

Listing Two

"C" source code for Stan Mitchell's program to dump the device driver chain.

```
/***********************************************************************
 * dd_dump.c              program for displaying device driver chain   *
 *                        for DOS 2.00,2.10,3.00                       *
 *                                                                     *
 * by stan mitchell       november 21,1984                             *
 *                        Lattice C v. 1.04                            *
 ***********************************************************************/

#define void  int
#define byte  char
#include "stdio.h"

```

(Continued on page 124)

16-Bit Toolbox (Listing Continued, text begins on page 118)

(Listing Continued, text begins on page 118)

Listing Two

```
/* DOS function calls */
#define OPENFCB    0x0f
#define CLOSEFCB   0x10
#define VERSION    0x30

extern  int _peek();

struct  DEVHDR
        {
        unsigned  nxthdr_off;  /* doubleword ptr to next device hdr in chain */
        unsigned  nxthdr_seg;
        unsigned  attr;        /* type of device driver */
        unsigned  strat;       /* device strategy entry point */
        unsigned  intrpt;      /* interrupt entry point */
        char      dname[8];    /* device name */
        };

struct  FCB
        {
        byte      drive;        /* drive designator */
        char      fname[11];    /* file or device name */
        unsigned  curblk;       /* current blk (set to 0 by OPENFCB) */
        unsigned  recsiz;       /* logical record size (set to 0x80 by OPENFCB) */
        long      fsize;        /* file size in bytes */
        unsigned  date;         /* creation or last update date */
        byte      sys_rsv[10];  /* fields reserved for DOS */
        byte      bset[5];      /* relative record numbers */
        };

struct  RSV2_X
        {
        unsigned  time;         /* creation or last update time */
        byte      attribute;    /* device or file attribute */
        unsigned  dhdr_off;     /* offset address of device header */
        unsigned  dhdr_seg;     /* segment address of device header */
        byte      unk[3];       /* unknown usage */
        };

struct  RSV3_X
        {
        unsigned  time;         /* creation or last update time */
        unsigned  attribute;    /* device or file attribute */
        unsigned  dhdr_off;     /* offset address of device header */
        unsigned  dhdr_seg;     /* segment address of device header */
        byte      unk[2];       /* unknown usage */
        };

struct  DEVHDR  device;          /* device header */
struct  FCB     device_fcb;      /* standard FCB for device */
struct  RSV2_X  *dos2x;          /* reserved field definitions DOS 2.x */
struct  RSV3_X  *dos3x;          /* reserved field definitions DOS 3.x */

main()
{
int      dev_off,dev_seg,i,j;

initfcb(0,"NUL        ");  /* set up standard FCB for NUL device */
if (bdos(OPENFCB,&device_fcb) & 0xff) /* open the device */
        {
        printf("Unable to open device\n");
        exit(1);
        }
if ((bdos(VERSION,0) & 0xff) == 3)
        {  /* the reserved fields are allocated differently in DOS 3.x */
        dos3x=(struct RSV3_X *) &device_fcb.sys_rsv[0];
        dev_off=dos3x->dhdr_off;
        dev_seg=dos3x->dhdr_seg;
        }
else if ((bdos(VERSION,0) & 0xff) == 2)
        {  /* ... than they were in DOS 2.x */
        dos2x=(struct RSV2_X *) &device_fcb.sys_rsv[0];
        dev_off=dos2x->dhdr_off;
        dev_seg=dos2x->dhdr_seg;
```

```
          }
    else
          { /* forget it for DOS 1.10 or earlier */
          printf ("  DOS 2.00 or newer required\n");
          exit(1);
          }

    printf("   Device driver chain .... \n");
    printf("\n");

     /* display the current DOS chain of device drivers */
    printf("   ptr      type    name       strategy ptr   interrupt ptr  \n");

    for (;;)
       {
       printf("%04x:%04x  ",dev_seg,dev_off); /* device header ptr */
       if (dev_off == 0xffff) break;          /* if last in chain, offset=0xffff */
       /* move the device header into the data segment structure "device" */
       _peek(dev_seg,dev_off,&device,sizeof(device));
       printf(" %04x  ",device.attr);         /* display the device attribute word */
       if ((device.attr & 0x8000) == 0)       /* if a block device ... */
            for (j=0;j<=3;j++) printf("%02x",device.dname[j]); /* there is no device name */
       else
            for (j=0;j<=7;j++) printf("%c",device.dname[j]); /* otherwise display device name */

       printf("        %04x:%04x       ",dev_seg,device.strat); /* strategy entry point */
       printf("%04x:%04x    \n",dev_seg,device.intrpt); /* "interrupt" entry point */
       dev_seg=device.nxthdr_seg; /* set up ptr to next device header */
       dev_off=device.nxthdr_off; /*   and loop back to display it */
       }

    }

    /* initialize standard FCB */
    void     initfcb(drv,name)
    byte     drv;
    char     name[];
     {
     int     i;

     device_fcb.drive=drv;  /* drive designation: default drive=0, A=1, etc. */
     for (i=0;i<=10;i++) device_fcb.fname[i]=name[i]; /* device or file name */
     for (i=0;i<=4;i++) device_fcb.bset[i]=0; /* fields not zeroed by open call */
     }
```

End Listing Two

Listing Three

Assembly language source for " peek" library function, used in Stan Mitchell's device driver dump program.

```
                                PAGE
                                PAGE    62,132
                                NAME    peek
                        ;************************************************************************
                        ;
                        ;       void    _peek(segment,offset,buffer,nbytes)
                        ;
                        ;       unsigned  segment;  /* segment portion of memory addr */
                        ;       unsigned  offset;   /* offset portion of memory addr */
                        ;       byte      *buffer;  /* local memory buffer (in data segment) */
                        ;       unsigned  nbytes;   /* number of bytes to transfer */
                        ;
                        ;        Lattice C assembly language interface convention followed
                        ;           (ES=DS on entry)
                        ;************************************************************************
                        ;
 = 0004                 EXTRA   EQU    4
                        ;
                        PGROUP  GROUP   PROG
 0000                   PROG    SEGMENT BYTE  PUBLIC  'PROG'
                                PUBLIC  _PEEK
                                ASSUME  CS:PGROUP
                        ;
```

(Continued on next page)

Listing Three

```
0000                 _PEEK  PROC    NEAR
0000  55                    PUSH    BP            ;
0001  8B EC                 MOV     BP,SP         ;
0003  1E                    PUSH    DS            ;
0004  56                    PUSH    SI            ;
0005  57                    PUSH    DI            ;
0006  51                    PUSH    CX            ;
                            ;
                            ;get source segment
0007  8E 5E 04              MOV     DS,WORD PTR [BP+EXTRA]
                            ;
                            ;get source offset
000A  8B 76 06              MOV     SI,WORD PTR [BP+EXTRA+2]
                            ;
                            ;get destination offset in ES
000D  8B 7E 08              MOV     DI,WORD PTR [BP+EXTRA+4]
                            ;
                            ;get byte count for transfer
0010  8B 4E 0A              MOV     CX,WORD PTR [BP+EXTRA+6]
                            ;
                            ;move the bytes into local buffer
0013  F3/ A4                REP     MOVSB
                            ;
0015  59                    POP     CX            ;
0016  5F                    POP     DI            ;
0017  5E                    POP     SI            ;
0018  1F                    POP     DS            ;
0019  5D                    POP     BP            ;
001A  C3                    RET                   ;
001B                 _PEEK  ENDP
001B                 PROG   ENDS
                            END
```

End Listing Three

Listing Four

```
Gregor Owen's Microsoft Assembler Bug of the Month.
Assemble this brief program for some entertaining error messages,
then comment out the line "es:lodsb" and assemble it again.


microwiff segment

          assume cs:microwiff,ds:microwiff,es:microwiff,ss:microwiff

microwimp:

          es:lodsb
          lods      byte ptr es:[si+0]
          mov       ax,es:[si+1]

microwiff ends

          end       microwimp
```

End Listings

ADVERTISER INDEX

Advertising Sales Offices

East Coast
Walter Andrzejewski (415) 424-0600

Midwest/West Central
Michele Beaty (317) 875-0557

Northern California/Northwest
Shawn Horst (415) 424-0600

Southern California/Southwest
Beth Dudas (714) 643-9439

Classified Advertising
Alice Abrams (415) 424-0600

Advertising Director
Stephen Friedman (415) 424-0600

Advertising Coordinator
Lisa Boudreau (415) 424-0600

Assistant Advertising Coordinator
Jay Horvath (415) 424-0600

Introducing the MIX Editor

(with Split Screen - both horizontal and vertical)

A Powerful Addition To Any Programmer's Tool Box

Features	Offer	More features
Full Screen Editing	*Introductory Offer Only*	Programmable
WordStar Key Layout		Macro Commands
Custom Key Layouts	**29.95**	Custom Setup Files
Terminal Configuration		Mnemonic Command Mode
Help Files		Multiple File Editing
Backup Files	30 Day Money Back Guarantee	Split Screen Editing

For PCDOS/MSDOS (2.0 and above/128K) • IBM PC/Compatibles, PC Jr., Tandy 1000/1200/2000, & others
For CPM80 2.2/3.0 (Z80 required/64K) • 8" SSSD, Kaypro 2/4, Osborne I SD/DD, Apple II, & others

Great For All Languages

A general purpose text processor, the MIX Editor is packed with features that make it useful with any language. It has auto indent for structured languages like Pascal or C. It has automatic line numbering for BASIC (255 character lines). It even has fill and justify for English.

Terminal Configuration

A utility for defining terminal features (smart features included) allows the editor to work with any terminal. Over 30 of the most popular terminals are built-in.

Custom Key Layouts

Commands are mapped to keys just like WordStar. If you don't like the WordStar layout, simply change it. Any key can be mapped to any command. You can also define a key to generate a string of characters, great for entering keywords.

Split Screen

You can split the screen horizontally or vertically and edit two files simultaneously.

Macro Commands

The MIX Editor allows a sequence of commands to be executed with a single keystroke. You can define a complete editing operation and perform it at the touch of a key.

MIX software
2116 E. Arapaho
Suite 363
Richardson, Tx 75081
(214) 783-6001

MSDOS is a trademark of Microsoft
PCDOS is a trademark of IBM
CPM80 is a trademark of Digital Research
WordStar is a trademark of MicroPro

Custom Setup Files

Custom keyboard layouts and macro commands can be saved in setup files. You can create a different setup file for each language you use. The editor automatically configures itself using a setup file.

Command Mode

Command mode allows any editor command to be executed by name. It is much easier to remember a command name versus a complicated key sequence. Command mode makes it easy to master the full capability of the editor. Frequently used commands can be mapped to keys. Infrequent commands can be executed by name.

Editor Commands

The editor contains more than 100 commands. With so many commands, you might think it would be difficult to use. Not so, it is actually extremely simple to use. With command mode, the power is there if you need it, but it doesn't get in your way if you don't. Following is a list of some of the commands.

Cursor Commands

Left/Right/Up/Down
Tab Right/Tab Left
Forward Word/Backward Word
Beginning of Line/End of Line
Scroll Up/Scroll Down
Window Up/Window Down
Scroll Left/Scroll Right
Top of File/Bottom of File
• • •

Block Commands

Copy/Move/Delete
Read/Write
Lower Case/Upper Case
Fill/Justify
Print

File Commands

Directory (with wild cards)
Show File/Help File
Input/Output File
Delete File/Save File

Other Commands

Split Screen/Other Window
Find String/Replace String
Replace Global/Query Replace
Delete Line/Undelete Line
Delete Word/Undelete Word
Insert Mode/Overwrite Mode
Open Line/Join Line
Duplicate Line/Center Line
Set Tab/Clear Tab
• • •

To Order: **Call Toll Free 1-800-622-4070,** (Illinois only **1-800-942-7317**)
Mix Editor ____ $29.95 + shipping ($5 USA/$10 Foreign) Texas residents add 6% sales tax

Visa ____ MasterCard ____ Card # ____________ Exp. Date ____________
COD ____ Check ____ Money Order ____ Disk Format ____________
Computer ____________ Operating System: MSDOS ____ PCDOS ____ CPM80 ____
Name ____________
Street ____________
City/State/Zip ____________
Country ____________
Phone ____________

MIX software
2116 E. Arapaho
Suite 363
Richardson, Tx 75081
Dealer Inquiries Welcome
Call (214) 783-6001

MOST SIGNIFICANT PRODUCT OF THE YEAR - PC WEEK

They said it couldn't be done. Borland Did It. Turbo Pascal 3.0

The industry standard

With more than 250,000 users worldwide Turbo Pascal is the industry's de facto standard. Turbo Pascal is praised by more engineers, hobbyists, students and professional programmers than any other development environment in the history of microcomputing. And yet, Turbo Pascal is simple and fun to use!

	TURBO 3.0	TURBO 2.0	MS PASCAL
COMPILATION SPEED	8.1	16	206
EXECUTION SPEED	9 SEC	13 SEC	20 SEC
CODE SIZE	12 K	12 K	35 K
BUILT-IN INTERACTIVE EDITOR	YES	YES	NO
ONE STEP COMPILE (NO LINKING NECESSARY)	YES	YES	NO
COMPILER SIZE	39K	35K	300K+
TURTLE GRAPHICS	YES	NO	NO
BCD OPTION	YES	NO	YES
PRICE	$69.95	$54.95	$295.00

(*) **Benchmark** run on an IBM PC using MS Pascal version 3.2 and the DOS linker version 2.6. The 179 line program used is the "Gauss-Seidel" program out of Alan R. Miller's book: *Pascal programs for scientists and engineers* (Sybex, page 128) with a 3 dimensional non-singular matrix and a relaxation coefficient of 1.0.

The best just got better: Introducing Turbo Pascal 3.0

We just added a whole range of exciting new features to Turbo Pascal:

- First, the world's fastest Pascal compiler just got faster. Turbo Pascal 3.0 (16 bit version) compiles twice as fast as Turbo Pascal 2.0! No kidding.
- Then, we totally rewrote the file I/O system, and we also now support I/O redirection.
- For the IBM PC versions, we've even added "turtle graphics" and full tree directory support.
- For all 16 Bit versions, we now offer two additional options: 8087 math coprocessor support for intensive calculations and Binary Coded Decimals (BCD) for business applications.
- And much much more.

The Critics' Choice.

Jeff Duntemann, PC Magazine: *"Language deal of the century . . . Turbo Pascal: It introduces a new programming environment and runs like magic."*

Dave Garland, Popular Computing: *"Most Pascal compilers barely fit on a disk, but Turbo Pascal packs an editor, compiler, linker, and run-time library into just 39K bytes of random-access memory."*

Jerry Pournelle, BYTE: *"What I think the computer industry is headed for: well documented, standard, plenty of good features, and a reasonable price."*

Portability.

Turbo Pascal is available today for most computers running PC DOS, MS DOS, CP/M 80 or CP/M 86. A XENIX version of Turbo Pascal will soon be announced, and before the end of the year, Turbo Pascal will be running on most 68000 based microcomputers.

An Offer You Can't Refuse.

Until June 1st, 1985, you can get Turbo Pascal 3.0 for only $69.95. Turbo Pascal 3.0, equipped with either the BCD or 8087 options, is available for an additional $39.95 or Turbo Pascal 3.0 with both options for only $124.95. As a matter of fact, if you own a 16-Bit computer and are serious about programming, you might as well get both options right away and save almost $25.

Update policy.

As always, our first commitment is to our customers. You built Borland and we will always honor your support.

So, to make your upgrade to the exciting new version of Turbo Pascal 3.0 easy, we will accept your original Turbo Pascal *disk* (in a bend-proof container) for a trade-in credit of $39.95 and your Turbo87 original *disk* for $59.95. This trade-in credit may only be applied toward the purchase of Turbo Pascal 3.0 and its additional BCD and 8087 options (trade-in offer is only valid directly through Borland and until June 1st, 1985).

NOT COPY-PROTECTED

TURBO PASCAL

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133

Carefully Describe your Computer System!

Mine is: ___ 8 bit ___ 16 bit
I Use: ___ PC-DOS ___ MS-DOS ___ CP/M 80 ___ CP/M 86
My computers' name/model is: ________

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

Name: ________
Shipping Address: ________
City: ________ State: ________ Zip: ________
Telephone: ________

For update: original Turbo disk must accompany order

YES! I want the Best! Please send: Quantity
Pascal 3.0 $ 69.95 ____
Pascal w/8087 $109.90 ____
Pascal w/BCD $109.90 ____
Pascal w/8087 & BCD $124.95 (SAVE $24.90) ____

*These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

Subtotal (CA 6% tax) ____
Trade-in Credit Claimed: ____
Amount Enclosed: ____
Payment: VISA MC BankDraft Check
Credit Card Expir. Date: ____/____
Card #: ________

COD's and Purchase Orders WILL NOT be accepted by Borland. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in US dollars drawn on a US bank.

Circle **no. 14** on reader service card.

BORLAND INTERNATIONAL

Software's Newest Direction
4585 Scotts Valley Drive
Scotts Valley, CA 95066
TELEX 172373